

Monte Carlo Methods in Vision, Graphics and Learning

Adrian Barbu, Song-Chun Zhu

April 11, 2017

¹Adrian Barbu is an associate professor of Statistics at Florida State University, Tallahassee, FL 32306. Song-Chun Zhu is a professor of Computer Science and Statistics at University of California, Los Angeles, CA 90095. Emails: abarbu@stat.fsu.edu; sczhu@stat.ucla.edu

Contents

1	Introduction to Monte Carlo Methods	13
1.1	Motivation and Objectives	13
1.2	Tasks in Monte Carlo computing	14
1.2.1	Task 1: Sampling and Simulation	15
1.2.2	Task 2: Estimating Quantities by Monte Carlo Simulation	17
1.2.3	Task 3: Optimization and Bayesian Inference	18
1.2.4	Task 4: Learning and Model Estimation	20
1.2.5	Task 5: Visualizing the Landscape	21
2	Sequential Monte Carlo	23
2.1	Sampling a 1-dimensional density	23
2.2	Importance Sampling and Weighted Samples	24
2.3	Sequential Importance Sampling(SIS)	27
2.3.1	Application: the number of self-avoiding walks	27
2.3.2	Application: particle filtering for tracking objects in video	30
2.3.3	Summary of the SMC Framework	33
2.4	Application: Ray Tracing by SMC	34
2.4.1	Example: glossy highlights	35
2.5	Preserving Sample Diversity in Importance Sampling	36
2.6	Monte Carlo Tree Search	39
3	Markov Chain Monte Carlo - the Basics	41
3.1	Topology of Transition Matrix: Communication and Period	43
3.2	The Perron-Frobenius Theorem	46
3.3	Convergence Measures	47
3.4	Markov Chains in Continuous or Heterogeneous State spaces	49
3.5	Ergodicity Theorem	50
3.6	MCMC for Optimization by Simulated Annealing	51
4	Metropolis Methods and Variants	53
4.1	The Metropolis-Hastings Algorithm	53
4.1.1	The original Metropolis-Hastings algorithm	54
4.1.2	Another version of the Metropolis-Hastings algorithm	55
4.1.3	Other acceptance probability designs	55
4.1.4	Key issues in Metropolis design	56
4.2	The Independence Metropolis Sampler	56

4.2.1	The eigenstructure of the IMS	57
4.2.2	General first hitting time for finite spaces	58
4.2.3	Hitting time analysis for the IMS	58
4.3	Reversible Jumps and Trans-Dimensional MCMC	59
4.3.1	Reversible Jumps	59
4.3.2	Toy Example: 1D Range image segmentation	60
4.4	Application: Counting People	64
4.4.1	Marked point process model	64
4.4.2	Inference by MCMC	65
4.4.3	Results	65
4.5	Application: Furniture Arrangement	66
5	Gibbs Sampler and its Variants	71
5.1	Gibbs Sampler	71
5.1.1	A major problem with the Gibbs sampler	72
5.2	Gibbs Sampler generalizations	74
5.2.1	Hit-and-Run	74
5.2.2	Generalized Gibbs Sampler	74
5.2.3	Generalized Hit-and-Run	75
5.2.4	Sampling with auxiliary variables	75
5.2.5	Simulated Tempering	76
5.2.6	Slice Sampling	76
5.2.7	Data Augmentation	77
5.2.8	Metropolized Gibbs Sampler	78
5.3	Data association and data augmentation	80
5.4	Julesz ensemble and MCMC sampling of texture	81
5.4.1	Image features and statistics	81
5.4.2	The Julesz ensemble - a mathematical definition of texture	82
5.4.3	The Gibbs ensemble and ensemble equivalence	84
5.4.4	Sampling the Julesz ensemble	85
5.4.5	Experiment: sampling the Julesz ensemble	86
6	Cluster Sampling Methods	89
6.1	Introduction: Potts model and Swendsen-Wang	89
6.2	Interpretations of the SW Algorithm	92
6.2.1	Interpretation 1: Metropolis-Hastings perspective	92
6.2.2	Interpretation 2: data augmentation	95
6.3	Some theoretical results	96
6.4	Swendsen-Wang Cuts for Arbitrary Probabilities	98
6.4.1	Step 1: data-driven clustering	99
6.4.2	Step 2: flipping of color	100
6.4.3	Step 3: accepting the flipping	100
6.4.4	Complexity Analysis	102
6.4.5	Example: SWC for Gaussian Mixture Models	102
6.5	Variants of the cluster sampling method	103
6.5.1	Cluster Gibbs sampling — the "hit-and-run" perspective	103
6.5.2	The multiple flipping scheme	104

6.6	Application: Image Segmentation	106
6.7	Multigrid and multi-level SW-cut	108
6.7.1	SW-cuts at multigrid	109
6.7.2	SW-cuts at multi-level	111
6.8	Subspace Clustering	112
6.8.1	Subspace Clustering by Swendsen-Wang Cuts	113
6.8.2	Application: Sparse Motion Segmentation	115
6.9	C4: Clustering Cooperative and Competitive Constraints	120
6.9.1	Overview of the Major Concepts of C^4	123
6.9.2	Graphs, Coupling, and Clustering	124
6.9.3	C^4 algorithm on flat graphs	129
6.9.4	Experiments on Flat Graphs	132
6.9.5	Checkerboard Ising Model	133
6.9.6	C^4 on Hierarchical Graphs	137
6.9.7	Experiments on Hierarchical C^4	140
7	Convergence Analysis of MCMC	143
7.1	Practical Methods for Monitoring	144
7.2	Coupling Methods for Card shuffling	145
7.2.1	Shuffling to the top	146
7.2.2	Riffle shuffling	146
7.3	Geometric Bounds, Bottleneck and Conductance	147
7.3.1	Geometric convergence	147
7.4	Peskun's Ordering and Ergodicity Theorem	149
7.5	Path Coupling and Exact Sampling	149
7.5.1	Coupling from the past	151
7.5.2	Application: sampling the Ising model	152
8	Data Driven Markov Chain Monte Carlo	155
8.1	Problem Formulation and Image Models	156
8.1.1	The Bayesian Formulation for Segmentation	156
8.1.2	The Prior Probability	157
8.1.3	The Likelihood for Grey Level Images	158
8.1.4	Model calibration	159
8.1.5	Image models for color	160
8.2	Anatomy of Solution Space	161
8.3	Exploring the Solution Space by Ergodic Markov chains	162
8.3.1	Three basic criteria for Markov chain design.	162
8.3.2	Five Markov chain dynamics	163
8.3.3	The bottlenecks	164
8.4	Data-Driven Methods	165
8.4.1	Method I: clustering in atomic spaces	165
8.4.2	Method II: Edge detection	168
8.5	Computing importance proposal probabilities	169
8.6	Computing Multiple Distinct Solutions	173
8.6.1	Motivation and a mathematical principle	173
8.6.2	A K -adventurers algorithm for multiple solutions	174

8.7	Image Segmentation Experiments	174
8.8	Application: Image Parsing	177
8.8.1	Bottom-Up and Top-Down Processing	180
8.8.2	Generative and Discriminative Methods	180
8.8.3	Markov Chain kernels and sub-kernels	181
8.8.4	DDMCMC and Proposal Probabilities	183
8.8.5	The Markov Chain Kernels	191
8.8.6	Image Parsing Experiments	198
9	Hamiltonian Monte Carlo	203
9.1	Introduction to Hamiltonian Mechanics	203
9.2	Properties of Hamiltonian Mechanics	204
9.2.1	Conservation of Energy	204
9.2.2	Reversibility	205
9.2.3	Symplectic Structure and Volume Preservation	205
9.3	The Leapfrog Discretization of Hamilton's Equations	206
9.3.1	Euler's Method and Modified Euler's Method	206
9.3.2	The Leapfrog Integrator	207
9.3.3	Properties of the Leapfrog Integrator	207
9.4	Hamiltonian Monte Carlo and Langevin Monte Carlo	208
9.4.1	Formulation of HMC	208
9.4.2	The HMC Algorithm	209
9.4.3	The LMC Algorithm	211
9.4.4	Proof of Detailed Balance for HMC	213
9.4.5	Tuning Standard HMC	214
9.5	Riemann Manifold HMC	215
9.5.1	Linear Transformations in HMC	215
9.5.2	RMHMC Dynamics	217
9.5.3	RMHMC Algorithm and Variants	219
9.5.4	Covariance Functions in RMHMC	220
9.6	HMC in Practice	221
9.6.1	Simulated Experiments on Constrained Normal Distributions	221
9.6.2	Sampling Logistic Regression Coefficients with RMHMC	224
9.6.3	The Alternating Backward Propagation (ABP) Algorithm	227
10	Stochastic Gradient for Learning	231
10.1	The Robbins-Monro Algorithm	231
10.2	Parameter estimation methods for Gibbs/MRF models	233
10.2.1	Learning in Gibbsian fields – a common framework	233
10.2.2	Three new algorithms	236
10.2.3	Experiments	238
11	Mapping the Energy Landscape	243
11.1	ELM construction	246
11.1.1	Space partition	246
11.1.2	Generalized Wang-Landau algorithm	246
11.1.3	Constructing the ELM	248

11.1.4	Estimating the mass and volume of nodes in the ELM	249
11.1.5	Characterizing the difficulty (or complexity) of learning tasks	250
11.1.6	MCMC moves in the model space	251
11.1.7	ELM convergence analysis	252
11.2	Experiment I: ELMs of Gaussian Mixture Models	252
11.2.1	Energy and Gradient Computations	253
11.2.2	Bounding the GMM space	254
11.2.3	Experiments on Synthetic Data	255
11.2.4	Experiments on Real Data	257
11.3	Experiment II: ELM of Bernoulli Templates	258
11.3.1	Experiments on Synthetic Data	259
11.3.2	Experiments on Real Data	260
11.4	Experiment III: ELM of bi-clustering	261
12	Curriculum Learning	269
12.1	Learning Dependency Grammars	269
12.1.1	Energy Function	270
12.1.2	Discretization of the hypothesis space	270
12.1.3	Experiments	271

Preface

Real world systems studied in sciences (e.g. physics, chemistry, and biology) and engineering (e.g. vision, graphics and robotics) involve complex interactions between large numbers of components. The representations for such systems are probabilistic models defined on graphs in high-dimensional spaces, to which analytic solutions are often unavailable. As a result, Monte Carlo methods have been used as a common tool for simulation, estimation, inference and learning in science and engineering. It is of no surprise that Monte Carlo method was ranked no.1 in the top-ten list of mostly used algorithms in the 20th century (Dongarra and Sullivan, 2000). With the ever-growing computing capacities, people are tackling more complex problems and adopting more advanced models. Monte Carlo methods will play an important role for sciences and engineering in the 21th century.

In history, several communities have contributed to the development of Monte Carlo methods.

- *Physics and chemistry*, for example, the early Metropolis algorithm (Metropolis, Rosenbluth, Rosenbluth, Teller and Teller, 1953), simulated annealing (Kirkpatrick, Gelatt, and Vecchi, 1983) and cluster sampling (Swendsen and Wang, 1987; Edwards and Sokal 1988), and the recent work on disconnectivity graph (Becker and Karpus, 1997) for visualizing the landscape of spin-glass models.
- *Probability and statistics*, for example, stochastic gradient (Robin and Monro, 1951, Younes 1988), Hastings dynamics (Hastings, 1970), data augmentation (Tanner and Wong, 1987), reversible jumps (Green, 1995), dynamic weighting (Wong and Liang, 1997) for studying bio-informatics, and numerous analysis for bounding the convergence of Markov chain Monte Carlo (Diaconis 1988, Diaconis and Stroock, 1991, and Liu, 1991).
- *Theoretical computer science*, for example, the convergence rate of clustering sampling by (Jerrum and Sinclair, 1989, Cooper and Frieze, 1999).
- *Computer vision and pattern theory*, for example, the Gibbs sampler (Geman and Geman, 1984) for image processing, Jump-diffusion (Miller and Grenander, 1994) for segmentation, the condensation algorithm for object tracking (Isard and Blake, 1996), to the recent work on Data-driven Markov chain Monte Carlo (Tu and Zhu, 2002) and generalized Swendsen-Wang cut (Barbu and Zhu, 2005) for image segmentation and parsing.

As these areas are very diverse and speak different languages, inter-disciplinary communication has been rare. This poses a big challenge for people who want to use Monte Carlo methods, especially people in computer science and engineering.

On one aspect, effective MC algorithms must explore the underlying structures of the problem, thus they are domain or problem specific and hardly accessible to outsiders. For example, many

important physics work, like (Swendsen and Wang, 1987), have only 2-3 pages without background introduction, and appears utterly mysterious to people in computer science and engineering.

On the other hand, general MC algorithm invented by statisticians are well-explained, but are usually found ineffective when they are implemented in generic ways by engineers without utilizing the structures of the underlying models and representations. As a result, there is a wide-spread misperception among engineers and graduate students that Monte Carlo methods are too slow and usually do not work. This is unfair to the Monte Carlo methods and unfortunate for the innocent students.

This book is written for researchers and graduate students in computer science and engineering. It covers all the interesting topics with both theoretical foundations and intuitive ideas developed in the four disciplines above, while leaving out small tricks which are less applicable or do not work in practice.

It illustrates the arts of Monte Carlo design using classical problems in computer vision, graphics and learning, and thus can be used as a reference book by researchers in these areas.

It can also be used as a textbook for teaching a graduate course in computer science and engineering. A draft has been used as a textbook in statistics at the University of California, Los Angeles.

The authors would like to thank many colleagues and friends, to mention a few names in alphabetic orders, Maria Pavlovskaja, Kewei Tu, Zhuowen Tu, Tianfu Wu, Yingnian Wu, Craig Yu, Qing Zhou for discussions and for allowing us to use their materials as examples in the book. The authors also like to acknowledge the support of a DARPA grant FA 8650-11-1-714, a MURI grant ONR N00014-10-1-0933, and two NSF grants IIS 1018751 and IIS-1423305.

Biosketches of the Authors

Adrian Barbu received his Ph.D. in Mathematics from Ohio State University in 2000 (advised by Dr. Avner Ash) and his Ph.D. in Computer Science from UCLA in 2005 (advised by Dr. Song-Chun Zhu). From 2005 to 2007 he was a research scientist and later project manager in Siemens Corporate Research, working in medical imaging. He received the 2011 Thomas A. Edison Patent Award with his co-authors from Siemens for their work on Marginal Space Learning. From 2007 he joined the Statistics department at Florida State University, first as assistant professor, and since 2013 as associate professor. His research interests are in computer vision, machine learning and medical imaging.



Song-Chun Zhu received his PhD degree in Computer Science from Harvard University in 1996 (advised by Dr. David Mumford). He is currently a professor of Statistics and Computer Science, and director of the Center for Vision, Learning, Cognition and Autonomy, at University of California, Los Angeles. Over the past 25 years, his research interest has been to pursue a common statistical framework for vision, and broadly intelligence. He proposed the Spatial, Temporal and Causal And-Or graph (STC-AOG) as a unified representation for modeling, inference and learning. He has published over 180 papers in computer vision, statistical learning, cognition and AI, and robot autonomy. He has received a number of honors, including the David Marr Prize in 2003 for image parsing with Z. Tu et al., the Marr Prize honorary nominations in 1999 for texture modeling and 2007 for object modeling with Y. Wu et al. As a junior faculty he received in 2001 the Sloan Fellow in Computer Science, NSF Career Award, and ONR Young Investigator Award. In 2008 he received the Aggarwal prize from the Intl Association of Pattern Recognition for “contributions to a unified foundation for visual pattern conceptualization, modeling, learning, and inference”. In 2013 he received the Helmholtz Test-of-time prize for a paper on image segmentation. He is a fellow of IEEE Computer Society since 2011. He has been the principal investigator leading several ONR MURI and DARPA teams working on scene and event understanding and cognitive robots under a unified mathematical framework.



Chapter 1

Introduction to Monte Carlo Methods

1.1 Motivation and Objectives

Monte Carlo, named after a casino in Monaco, stands for a business operation that simulates complex probabilistic events using simple random events – tossing dice. In Monte Carlo computing, people repeatedly call a pseudo-random number generator `rand()` which returns a real number in $[0, 1]$, and use them to generate *samples*, i.e. a population, as a fair representation of an arbitrary probability distribution (a.k.a the target probability) under study.

In general, Monte Carlo methods are divided in two categories:

- Sequential Monte Carlo methods, which preserve and propagate a population of examples by sequential sampling and importance reweighting, often in a low dimensional state space.
- Markov chain Monte Carlo method, which simulate Markov chains to explore the state space with its stationary probability designed to converge to a given target probability.

In engineering applications, e.g. computer vision, graphics and machine learning, the target functions are defined on graph representations, and people often face the choices of three types of modeling and computing paradigms that make trade-offs between model accuracy and computation complexity.

- Approximate model with exact computing. One simplifies the representation by breaking the loopy connections or removing certain energy terms. Once the underlying graph becomes a tree or a chain, then algorithms like Dynamic programming, are applicable to find the exact solution to the approximated problem. In the same class are problems for which a convex approximation of the energy is found and a convex optimization algorithm is used to find the global energy optimum. Examples include L_1 -penalized regression (lasso) [183] and classification, where the non-convex L_0 penalty on the number of nonzero model weights is replaced with the convex L_1 penalty.
- Exact model with local computing. One stays with the original representation and target function, but use approximate algorithm, e.g. gradient descent, to find a local solution and thus relies on heuristics to guide the initial states.
- Exact model with asymptotic global computing. This is the Monte Carlo methods, which simulate large enough samples over time, and converge to the globally optimal solution with high probability.

Monte Carlo methods have been used in many different tasks, which we shall elaborate with examples in the next section.

1. Simulating a system and its probability distribution $\pi(x)$

$$x \sim \pi(x); \quad (1.1)$$

2. Estimating a quantity through Monte Carlo integration

$$c = E_{\pi}(f(x)) = \int \pi(x) f(x) dx; \quad (1.2)$$

3. Optimizing a target function to find its modes (maxima or minima)

$$x^* = \arg \max \pi(x); \quad (1.3)$$

4. Learning parameters from a training set to optimize some loss functions. For example, the maximum likelihood estimation from a set of examples $\{x_i, i = 1, 2, \dots, M\}$

$$\Theta^* = \arg \max \sum_{i=1}^M \log p(x_i; \Theta); \text{ and} \quad (1.4)$$

5. Visualizing the energy landscape of a target function and thus quantifying the difficulty of one of the tasks above and the efficiency of various algorithms. For example in biology people are interested in the energy landscape of protein folding. Different proteins have different landscapes and local minima of the energy landscape could be related to certain diseases (e.g. Alzheimer's disease). In computer vision the energy landscape of learning algorithms such as Convolutional Neural Networks (CNN) are interesting to study to understand why they seem to give good results independent of the initialization (are all the local minima equivalent up to a permutation of the filters?), or for other learning algorithms to understand what are the difficulties of learning the correct model and how the energy landscape changes with the number of observations.

As one could see, Monte Carlo methods can be used for many complex problems.

1.2 Tasks in Monte Carlo computing

Real world systems studied in sciences (e.g. physics, chemistry, and biology) and engineering (e.g. vision, graphics, machine learning and robotics) involve complex interactions between large numbers of components. Such systems are often represented as graphs where the vertices represent components and the edges the interactions. The behavior of the system is governed by a probabilistic model defined on the graph.

For example, in statistical physics, ferro-magnetic materials are represented by the classical Ising and Potts models [165]. These models are also used in computer vision to represent the dependency between adjacent pixels in terms of Gibbs distributions and Markov random fields.

In general we are given a number of observations $\{\mathbf{x}_1, \dots, \mathbf{x}_n\} \sim f(\mathbf{x})$ that represent samples from the “true” probabilistic model $f(\mathbf{x})$. In reality, $f(\mathbf{x})$ is usually unknown and can only be approximated by the empirical samples $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$.

Many times we are interested in learning the unknown “true” model $f(\mathbf{x})$, which means approximating it with a parametric model $P(\mathbf{x}; \theta)$. In many occasions, learning the model or even finding how well the learned model $P(\mathbf{x}; \theta)$ compares to the true model means obtaining samples $\mathbf{x} \sim P(\mathbf{x}; \theta)$ from it and computing some sufficient statistics on these samples. Thus sampling is one of the fundamental tasks of Monte Carlo computing.

1.2.1 Task 1: Sampling and Simulation

We denote a 2D image lattice by

$$\Lambda = \{(i, j) : 1 \leq i, j \leq N\}. \quad (1.5)$$

Each pixel is a vertex with image intensity $\mathbf{I}_{(i,j)} \in \{0, \dots, 255\}$. An image, denoted by \mathbf{I}_Λ is a microscopic state of the underlying system governed by a probability $\pi(\mathbf{I}_\Lambda; \Theta)$. In other words, when the system reach a dynamic equilibrium, its state follow a Gibbs distribution

$$\mathbf{I}_\Lambda \sim \pi(\mathbf{I}_\Lambda; \Theta) \quad (1.6)$$

where Θ is a vector of K parameters, and the Gibbs distribution can be written in the following form,

$$\pi(\mathbf{I}_\Lambda; \Theta) = \frac{1}{Z} \exp\{-\langle \Theta, H(\mathbf{I}_\Lambda) \rangle\}. \quad (1.7)$$

In the above formulas, Z is a normalizing constant, $H(\mathbf{I}_\Lambda)$ is a vector of K sufficient statistics of image \mathbf{I}_Λ and the inner product is called the potential function $U(\mathbf{I}) = \langle \Theta, H(\mathbf{I}_\Lambda) \rangle$.

When the lattice is sufficiently large, the probability mass of $\pi(\mathbf{I}_\Lambda; \theta)$ will focus on a subspace, called the *micro-canonical ensemble* in statistical physics [116]

$$\Omega_\Lambda(\mathbf{h}) = \{\mathbf{I}_\Lambda : H(\mathbf{I}_\Lambda) = \mathbf{h}\}. \quad (1.8)$$

$\mathbf{h} = (h_1, \dots, h_k)$ is a constant vector called the macroscopic state of the system.

Therefore, drawing fair samples from the distribution $\Omega_\Lambda(\mathbf{h}) \sim \pi(\mathbf{I}_\Lambda; \Theta)$ is equivalent to sampling from the ensemble $\Omega_\Lambda(\mathbf{h}) \in \Omega_\Lambda(\mathbf{h})$. In plain language, the sampling process is to simulate the “typical” microscopic state of the system. In computer vision, this is often called *synthesis* – a way to verify the sufficiency of the underlying model.

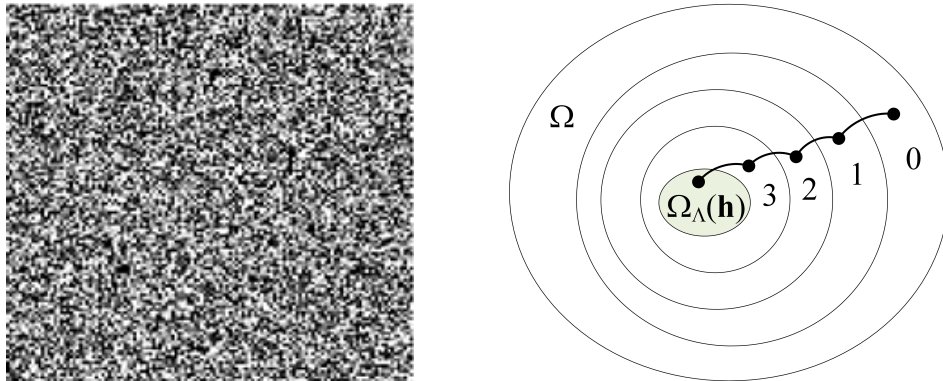


Figure 1.1: Left: A typical image sampled from a Gaussian model. Right: a set of nested ensemble spaces $\Omega_\Lambda(\mathbf{h})$ with increasing number of constraints from $K = 0, 1, 2, 3$.

Example 1.1. Simulating Gaussian noise images. In a large lattice, we define a “Gaussian noise” pattern as an ensemble of images with fixed mean and variance.

$$\text{Gaussian noise} = \Omega_{\Lambda}(\mu, \sigma^2) = \{\mathbf{I}_{\Lambda} : \frac{1}{N^2} \sum_{(i,j) \in \Lambda} I(i,j) = \mu, \frac{1}{N^2} \sum_{(i,j) \in \Lambda} (I(i,j) - \mu)^2 = \sigma^2\}.$$

In this case, the model has $K = 2$ sufficient statistics. Figure 1.1 displays a typical noise image as a sample from this ensemble or distribution.

Note

Why is that the highest probability image \mathbf{I}_{Λ} is not a typical image from $\Omega_{\Lambda}(\mu, \sigma^2)$?

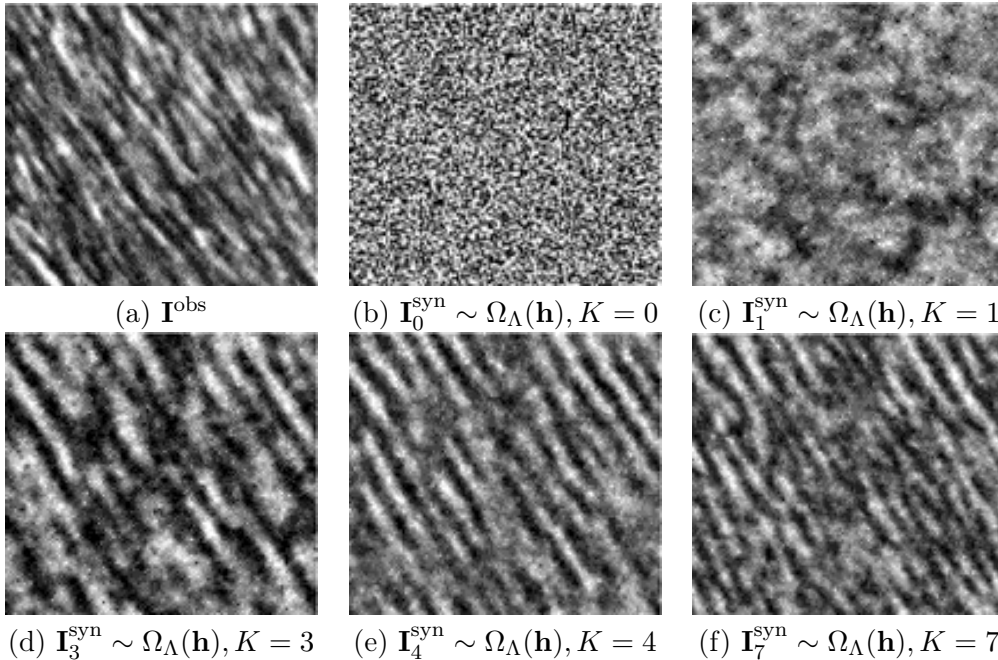


Figure 1.2: Simulating texture patterns from 5 different equivalence classes. Courtesy of Zhu, Wu and Mumford [222]

Example 1.2. Simulating texture patterns. As we will discuss in later section 5.4, each texture pattern is defined as an equivalence class,

$$\text{a texture} = \Omega_{\Lambda}(\mathbf{h}) = \{\mathbf{I}_{\Lambda} : H(\mathbf{I}_{\Lambda}) = \mathbf{h} = (h_1, \dots, h_K)\}. \quad (1.9)$$

In this example, the sufficient statistics $H_k(\mathbf{I}_{\Lambda}), k = 1, 2, \dots, K$ are the histograms of Gabor filters. That is, any two texture images will be perceptually equivalent if they share the same set of histogram of Gabor filters. More detailed discussions are referred to chapter 5.4 and references [208, 222].

Figure 1.2 displays an example for texture modeling and simulation, and demonstrates the power of the Markov chain Monte Carlo (MCMC) methods. Since the 1960s, a famous psycho-physicist Julesz studied texture perception, raised a classical problem which was called the *Julesz quest* later:

What are the set of features and statistics such that two texture images sharing the same feature statistics cannot be told apart in early vision?"

While the psychological interest is to find the sufficient statistics \mathbf{h} from an image \mathbf{I}_Λ , the Julesz quest posed a significant challenge technically: how do we generate fair samples for a given statistics \mathbf{h} . This was answered in the late 1990s by Zhu, Wu and Mumford using the Markov chain Monte Carlo method [222]. Figure 1.2.(a) is an observed texture image \mathbf{I}^{obs} , from which one can extract any sufficient statistics \mathbf{h} under consideration. To verify a statistics \mathbf{h} , one needs to draw typical samples drawn from the ensembles, or equivalently some Gibbs distributions, which satisfy the K feature statistics. Figure 1.2.(b-f) are examples for $K = 0, 1, 3, 4, 7$ respectively. Each statistics is a histogram of Gabor filtered responses pooled over all pixels, and is selected sequentially in a learning process [222]. As it demonstrated, with $K = 7$ selected statistics, the generated texture images $\mathbf{I}_7^{\text{syn}}$ is perceptually equivalent to the observed image \mathbf{I}^{obs} , i.e.

$$h_k(\mathbf{I}_7^{\text{syn}}) = h_k(\mathbf{I}^{\text{obs}}), \quad k = 1, 2, \dots, 7. \quad (1.10)$$

The MCMC method plays a key role in solving the Julesz quest.

1.2.2 Task 2: Estimating Quantities by Monte Carlo Simulation

In scientific computing, one common problem is to compute the integral of a function in a very high dimensional space Ω ,

$$c = \int_{\Omega} \pi(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}. \quad (1.11)$$

This is often estimated through Monte Carlo integration. By drawing M samples from $\pi(\mathbf{x})$,

$$x_1, x_2, \dots, x_M \sim \pi(\mathbf{x}),$$

one can estimate c by the sample mean

$$\bar{c} = \frac{1}{M} \sum_{i=1}^M f(\mathbf{x}_i). \quad (1.12)$$

This is often done by the sequential Monte Carlo method. We briefly discuss three examples in the following.

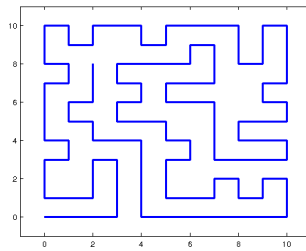


Figure 1.3: A self-avoiding walk of length 115.

Example 1.3. Approximate counting. In chemistry, an interesting problem is to calculate the number of polymers in a unit area. This is abstracted into a Self-Avoiding-Walks (SAW) problem in Monte Carlo computing. In an $N \times N$ lattice, a SAW \mathbf{r} is a path which does not go through any site twice. An example of a SAW is given in Figure 1.3. Denote the set of SAWs by

$$\Omega_{N^2} = \{\mathbf{r} : \text{SAW}(\mathbf{r}) = 1\}. \quad (1.13)$$

where $SAW()$ is a logic indicator. As we will discuss in Chapter 2, this quantity can be estimated by Monte Carlo integration,

$$c = \sum_{\mathbf{r} \in \Omega_{n^2}} 1 = \sum_{\mathbf{r} \in \Omega_{n^2}} \frac{1}{p(\mathbf{r})} p(\mathbf{r}) = E_p\left[\frac{1}{p(\mathbf{r})}\right] \approx \frac{1}{M} \sum_{i=1}^M \frac{1}{p(\mathbf{r}_i)}. \quad (1.14)$$

In the above formulas, the SAW paths are sampled from a reference model $p(\mathbf{r}_i)$ through random walks that grow the chain sequentially. For example, when $N = 10$, the estimated number of SAW paths starting from the lower-left corner $(0, 0)$ to the upper-right corner $(10, 10)$ is $(1.6 \pm 0.3) \times 10^{24}$. The true number is 1.56875×10^{24} .

Example 1.4. Particle filtering. In computer vision, a well-known task is tracking objects in a video sequence. Figure 1.4 is a simplified example, where the object (i.e. humans here) position is represented by the horizontal axis x , and each row is a video frame $\mathbf{I}(t)$ at time t . Given an input video $\mathbf{I}[0, t]$, the objective of online tracking is to approximately represent the posterior probability by a set of samples,

$$S(t) = \{(x_i(t), \omega_i(t)) : i = 1, 2, \dots, M\} \approx \pi(x(t) | \mathbf{I}[0, t]), \quad (1.15)$$

where $\omega_i(t)$ is the weight for $x_i(t)$. $S(t)$ encodes a non-parametric distribution as is illustrated by each row in Figure 1.4, and is propagated in time through the following recursive integration,

$$\pi(x(t+1) | \mathbf{I}[0, t+1]) = \int g(\mathbf{I}(t+1) | x(t+1)) p(x(t+1) | x(t)) \cdot \pi(x(t) | \mathbf{I}[0, t]) dx(t). \quad (1.16)$$

In this integration, $p(x(t+1) | x(t))$ is the dynamic model for object movement, and $g(\mathbf{I}(t+1) | x(t+1))$ is the image likelihood model measuring the fitness of position $x(t+1)$ to observation. Each sample in the set $S(t)$ is called a particle. By representing the whole posterior probability, the sample set $S(t)$ preserves the ambiguity for achieving robustness in object tracking.

Example 1.5. Monte Carlo Ray tracing. In computer graphics, Monte Carlo integration is used to implement the ray-tracing algorithm for image rendering. Given a three-dimensional physical scene with geometry, reflection, and illumination, the photons emit from light sources will bounce between object surfaces, or go through transparent objects before they hit the imaging plane. The ray tracing approach calculates the color and intensity of each pixel on the imaging plane by summation (integration) over all the lights tracing a ray from the origin that passing this pixel. This integration is computationally intensive and can be approximated by Monte Carlo method, as we will elaborate in Chapter 2.

1.2.3 Task 3: Optimization and Bayesian Inference

A basic assumption in computational vision, since Helmholtz (1860), is that biological and machine vision compute the most probable interpretation(s) from an input image. Denoting the interpretation by W , for the perceived world, one can pose it as an optimization problem that maximizes a Bayesian posterior probability,

$$W^* = \arg \max \pi(W | \mathbf{I}) = \arg \max p(\mathbf{I} | W) p(W), \quad (1.17)$$

where $p(W)$ is the prior model for how the real world scene may be organized, and $p(\mathbf{I} | W)$ is the likelihood for generating image \mathbf{I} from a given scene W .

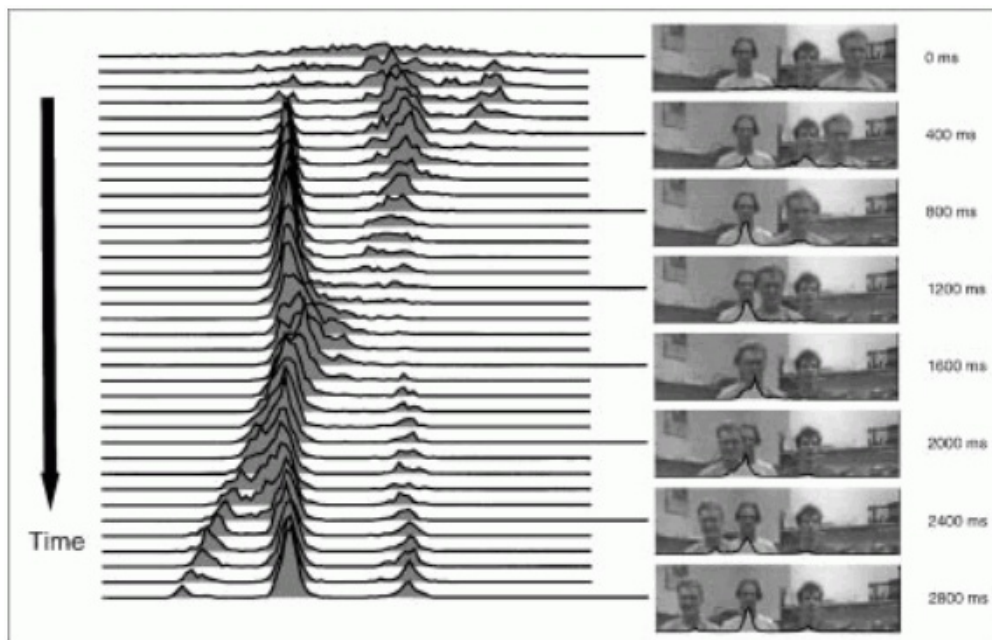


Figure 1.4: Tracking objects by Sequential Monte Carlo, courtesy of Isard and Blake [96].

Sometimes, images have multiple plausible interpretations, and thus in a more general setting, one needs to keep multiple distinct interpretations to approximately represent the posterior

$$\{(W_i, \omega_i) : i = 1, 2, \dots, M\} \approx p(W | \mathbf{I}). \quad (1.18)$$

Markov Chain Monte Carlo can be used to obtain samples from the posterior $p(W | \mathbf{I})$; however, sampling the posterior is not the same thing as maximizing it. The posterior can also be maximized by simulated annealing, which means sampling $p(W | \mathbf{I})^{1/T}$ where T is a parameter called temperature that is changed during the procedure. At the beginning of the annealing procedure, the temperature T is high, which means $p(W | \mathbf{I})^{1/T}$ is close to uniform and the MCMC can freely explore the solution space. During the annealing, the temperature T is slowly decreased according to an annealing schedule. As the temperature T decreases, the probability $p(W | \mathbf{I})^{1/T}$ becomes more and more peaked at the maximum locations and the MCMC explores these locations most. When the temperature is very small, the MCMC will be near a maximum of the posterior $p(W | \mathbf{I})$.

Example 1.6. Image segmentation and parsing. In computer vision, image segmentation and parsing is a core problem. In such tasks, since the underlying scene complexity is unknown, the number of variables in W is not fixed. Therefore the prior model $\pi(W)$ is distributed over a heterogeneous solution space, which is a union of subspaces of varying dimensions. When the objects in a scene is compositional, the solution W is a parse graph, and the structures of the solution space then become more complicated. Seeking optimal solutions in such complex space can be executed by Monte Carlo methods, which simulates Markov chains to traverse the solutions spaces by mixing a number of dynamics: death and birth, split and merge, model switching, and boundary diffusion. To improve computational efficiency, the Markov chains are guided by marginal distributions computing using data-driven approaches. We will elaborate the details in Chapter 8.

Figure 1.5 illustrates two instances computed by the Data-driven Markov Chain Monte Carlo method [193]. The left column shows the two input images, and segmentation results are in the

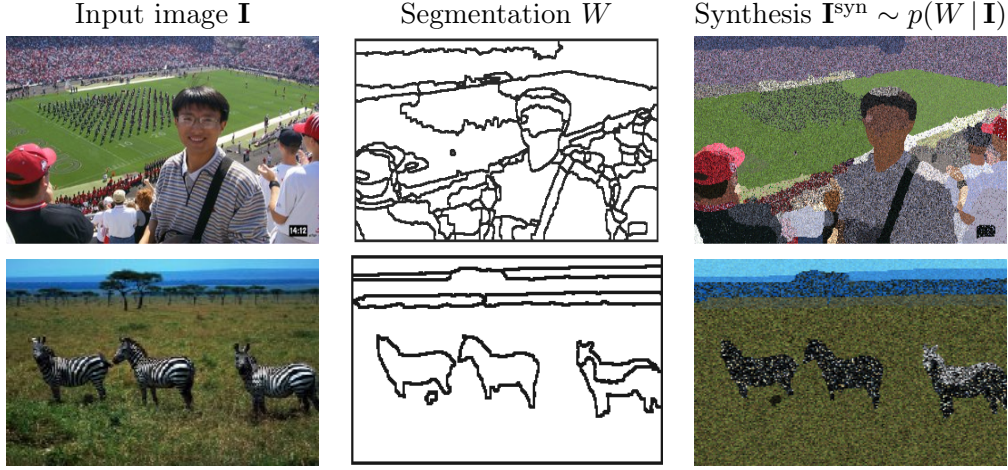


Figure 1.5: Image segmentation by Data-driven Markov chain Monte Carlo, courtesy of Tu and Zhu [193].

middle with each region being fitted to some likelihood model. To verify the world W^* computed by the computer algorithm, we sample typical images from the likelihood $\mathbf{I}^{\text{syn}} \sim p(W | \mathbf{I})$. In this example, the likelihood does not include face models, and thus the human face is not constructed.

1.2.4 Task 4: Learning and Model Estimation

In statistical learning and machine learning, one needs to compute parameters that optimize some loss functions, which are often highly non-convex, especially when hidden variables are involved. In the following, we briefly discuss two examples.

Example 1.7. Learning Gibbs distributions. Consider the Gibbs model that we mentioned in Example 1.2., we omit the lattice sign for clarity,

$$p(\mathbf{I}; \Theta) = \frac{1}{Z} \exp\{-\langle \Theta, H(\mathbf{I}) \rangle\}. \quad (1.19)$$

Given an set of examples $\{\mathbf{I}_i^{\text{obs}}, i = 1, 2, \dots, M\}$, the objective of learning is to estimate the parameters by maximizing the likelihood of the data,

$$\Theta^* = \arg \max \ell(\Theta), \text{ with } \ell(\Theta) = \sum_{i=1}^M \log p(\mathbf{I}_i^{\text{obs}}; \Theta). \quad (1.20)$$

The loss function $\ell(\Theta)$ is convex with respect to Θ . setting $\frac{\partial \ell}{\partial \Theta} = 0$, we derive the following constraint equations,

$$\int H(\mathbf{I}) p(\mathbf{I}; \Theta) d\mathbf{I} = \mathbf{h} = \frac{1}{M} \sum_{i=1}^M H(\mathbf{I}_i^{\text{obs}}). \quad (1.21)$$

This usually has to be solved by stochastic gradient. Let t denote the time step, one sample a set of typical examples $\{\mathbf{I}_i^{\text{syn}}, i = 1, 2, \dots, M\}$ from the current model $p(\mathbf{I}; \Theta(t))$ using Markov chain

Monte Carlo as in example 1.2, and use the sample mean $\hat{\mathbf{h}}(t) = \frac{1}{M} \sum_{i=1}^M H(\mathbf{I}_i^{\text{syn}})$ to estimate the expectation (i.e. Monte Carlo integration). The parameter is updated by gradient ascent,

$$\frac{d\Theta}{dt} = \eta(\mathbf{h} - \hat{\mathbf{h}}(t)), \quad (1.22)$$

where η is a step size.

The intuition is that the parameters Θ are updated so that the distribution on the observed data and the distribution obtained from the model cannot be told apart according to some sufficient statistics represented by $H(\mathbf{I})$.

Example 1.8. Restricted Boltzmann Machines. In deep learning, a Restricted Boltzmann machine (RBM) is a neural network with binary inputs and outputs. It has a matrix of weights (i.e. parameters) $W = (W_{ij})$ connecting a vector of visible units (inputs) \mathbf{v} with a vector of hidden units (outputs) \mathbf{h} . Note that this notation has different meaning from the \mathbf{h} in the previous example. It also has vectors \mathbf{a}, \mathbf{b} of biases for the visible units and hidden units respectively. The probability of a RBM is a Gibbs distribution

$$p(\mathbf{v}, \mathbf{h}; \Theta) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h}))$$

based on the RBM energy function

$$E(\mathbf{v}, \mathbf{h}; \Theta) = -\mathbf{a}^T \mathbf{v} - \mathbf{b}^T \mathbf{h} - \mathbf{v}^T W \mathbf{h}.$$

Training the RBM with a set of training examples $\mathbf{v}_1, \dots, \mathbf{v}_n$ usually means maximizing the log likelihood:

$$\Theta^* = (W, \mathbf{a}, \mathbf{b})^* = \operatorname{argmax} \sum_{i=1}^n \log \int p(\mathbf{v}_i, \mathbf{h}; \Theta) d\mathbf{h}$$

This optimization is done using Monte Carlo methods in the same way as the previous example. A variant method used in [94] is the so called by contrastive divergence.

1.2.5 Task 5: Visualizing the Landscape

In previous tasks, the Monte Carlo methods are used to draw fair examples from a target distribution (task 1), then use the samples to estimate quantities by Monte Carlo integration (task 2), and to optimize some posterior probability in the state space (task 3) or loss function in the model space (task 4). The most ambitious task that uses Monte Carlo methods is to visualize the whole energy landscape. This energy function can be the negative log-posterior probability $-\log p(W | \mathbf{I})$ on Ω_X for inference tasks, or the loss function $L(\Theta | \text{Data})$ in the parameter space for learning tasks.

In real world applications, these functions are highly non-convex with complex, often horrific, landscapes which are characterized by exponential number of local minima in high-dimensional spaces. Figure 1.6 illustrates a simplified two-dimensional energy function in a K-mean clustering and learning problem. This energy function has multiple local minima of varying depths and widths denoted by letters A, B, \dots, H . The red curves are level sets consisting of points on the same energy levels.

The objective of task 5 is to draw effective samples from the whole space use effective Markov chain Monte Carlo methods, and to map all the local minima in their energy basins, and to locate the saddle point connecting adjacency basins. The result is represented by a tree structure, which

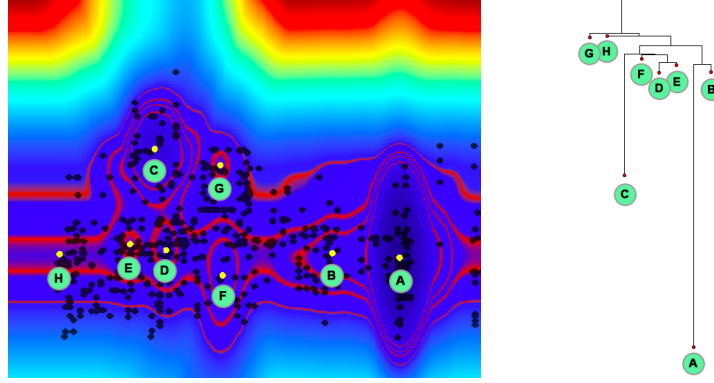


Figure 1.6: Visualizing the landscape. (left) An an energy function in 2D space. (right) the tree representation. Dark color means lower energy.

physicist called *disconnectivity graph* [12] when they map the landscapes of Spin-glass models. In this graph, each leaf node represents a local minimum and its depth represents the energy level. The energy level at which two adjacent leaf nodes meets is decided by their saddle point.

In the following, we show an example in the learning where the landscape is in the model space, not the state space, and thus is more difficult to compute.

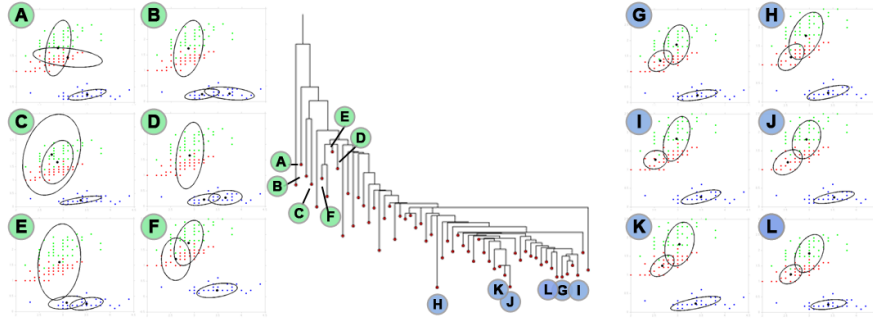


Figure 1.7: Visualizing the landscape of a clustering problem, from [157].

Example 1.9. Landscape of data clustering. K-mean clustering is a classical problem in statistical and machine learning. Given a finite number of points whose color indicates the true labels, the learning problem is to find the parameters Θ that best fit the data. Here Θ includes the means, variances, and weights of $K = 3$ Gaussian models. The energy function $\ell(\Theta)$ is a posterior probability with likelihood and a prior for Θ . In the literature, the popular algorithms are K-mean and EM algorithms which find only local minima. By exploring the models space where each point is a model Θ , one can visualize the landscape in Figure 1.7. The input data is from the Iris dataset in machine learning. twelve of the local minima A, B, \dots, L are illustrated on the two sides, where each Gaussian is an ellipse.

With this landscape, one can further visualize the behaviors of various algorithms, and quantize the intrinsic difficulties of the target function, either for inference or learning. One can also use it to study the key factors that affects the complexity of the landscape.

Chapter 2

Sequential Monte Carlo

Let $f(x)$ denote a probability distribution function and $\pi(x)$ denote a target probability distribution, based on a model. We want to find a model to make the target density function $\pi(x)$ converge to the true density function $f(x)$. In order to find this model, we may use a trial probability density, $g(x)$, which is known to us. In the most trivial case, we can choose that $g(x) = \text{Unif}[a, b]$. In this case, we can draw random samples with $x \sim g(x)$, *e.g.* $x = \text{rand}() \in \text{Unif}[0, 1]$.

2.1 Sampling a 1-dimensional density

Suppose $f(x) : \mathbb{R} \rightarrow \mathbb{R}$ is a one dimensional probability density function (pdf). Then the cumulative density function (cdf) $F(x) : \mathbb{R} \rightarrow [0, 1]$ is defined as

$$F(x) \stackrel{\text{def}}{=} \int_{-\infty}^x f(x) dx$$

We can obtain samples from the pdf $f(x)$ by taking uniform samples u back through the cdf $F(x)$ as $x = F^{-1}(u)$. More exactly we have

Lemma 2.1. *Suppose $U \sim \text{Unif}[0, 1]$ and F is the cdf of a one dimensional pdf f . Then, $X = F^{-1}(U)$ has the distribution f . Here we define $F^{-1}(u) = \inf\{x : F(x) \geq u\}$.*

Proof.

$$P(X \leq x) = P(F^{-1}(u) \leq x) = P(U \leq F(x)) = F(x) = \int_{-\infty}^x f(x) dx.$$

□

By definition, we know that $\frac{du}{dx} = \frac{dF(x)}{dx} = f(x)$, thus $P(x \in (x_0, x_0 + dx)) = P(u \in (u_0, u_0 + du)) = f(x) \cdot dx$.

In higher dimensional space, as long as all the data sequence can be quantized/ordered, then $f(x)$ will be transferred to a one dimensional problem. However, we typically do not use this method when $d \geq 3$, since the computation increases exponentially.

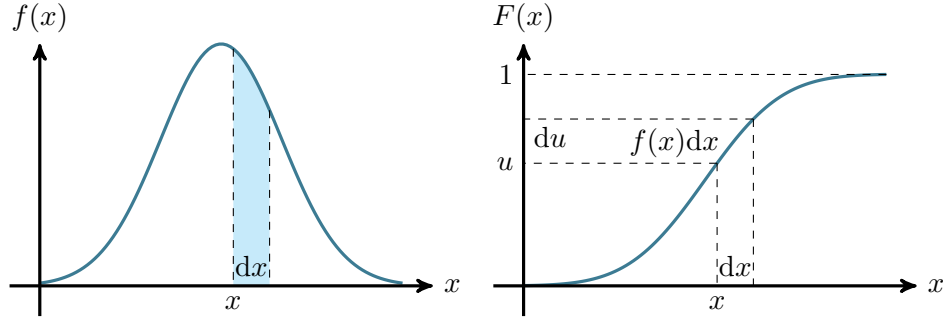


Figure 2.1: Left: A pdf $f(x)$. Right: Its corresponding cdf $F(x)$. The shaded region on the left has area $f(x)dx = du$

2.2 Importance Sampling and Weighted Samples

Suppose that we want to estimate a quantity

$$C = \int_{\Omega} \pi(x) \cdot h(x) dx = E_{\pi}[h(x)] \quad (2.1)$$

where $\pi(x)$ is a probability density function.

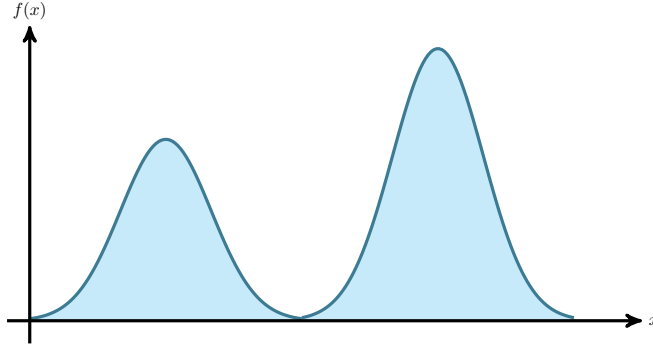


Figure 2.2: A multi-modal pdf $f(x)$.

If we can draw samples from $\pi(x)$, $D = \{x_1, x_2, \dots, x_n\} \sim \pi(x)$, then θ can be easily estimated by

$$\hat{C} = \frac{1}{n} \sum_{i=1}^n h(x_i)$$

Since the information of $\pi(x)$ is inherited in D , we do not need to write it in the formula.

However, if it is hard to draw samples directly from $\pi(x)$, we could draw samples from a simpler and easier trial distribution $D' = \{x'_1, x'_2, \dots, x'_n\} \sim g(x)$.

Then eq.(2.1) can be expressed as

$$C = \int_{\Omega} \pi(x) \cdot h(x) dx = \int_{\Omega} g(x) \cdot \left[\frac{\pi(x)}{g(x)} \cdot h(x) \right] dx \quad (2.2)$$

Suppose that the ratio $\frac{\pi(x)}{g(x)} \cdot h(x)$ is computable, we will have the estimation of C

$$\hat{C} = \frac{1}{n} \sum_{i=1}^n \frac{\pi(x'_i)}{g(x'_i)} \cdot h(x'_i) = \frac{1}{n} \sum_{i=1}^n \omega(x'_i) h(x'_i) \quad (2.3)$$

where $\omega(\cdot)$ is the weight.

Note

In Equ. (2.3), the weights $\{\omega(x_i), i = 1, 2, \dots, m\}$ depend on $g(x_i)$ in the denominator. Thus we cannot let $g(x) = 0$ whenever $\pi(x) \neq 0$.

Suppose that $\pi(x) = \frac{1}{Z} \exp(-E(x)/T)$, where Z is the normalizing constant but not computable. So $\pi(x)$ is represented by a weighted sample, $\{(x^{(i)}, \omega^{(i)}, i = 1, \dots, m)\}$.

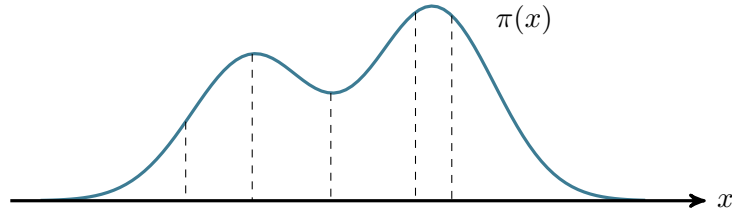


Figure 2.3: A pdf $\pi(x)$ is approximated by a weighted sample.

A special case: if

$$g(x) = \text{Unif}[a, b] = \begin{cases} \frac{1}{b-a} & x \in [a, b] \\ 0 & \text{otherwise.} \end{cases}$$

then

$$\hat{C} = \frac{1}{n} \sum_{i=1}^n \frac{\pi(x'_i)}{g(x'_i)} \cdot h(x'_i) = \frac{b-a}{n} \sum_{i=1}^n \pi(x'_i) \cdot h(x'_i)$$

In general, we will have the following three scenarios

- 1) We draw uniform samples, but the weights are different, *i.e.* we draw from Uniform distribution
- 2) We assign equal weight to all the samples, but the frequency are different, *i.e.* we draw from $\pi(x)$.
- 3) We draw samples from a easier distribution $g(x)$, which is an approximation of $\pi(x)$.

It is easy to understand that $\text{Unif}() \ll g() \ll \pi()$, here “ \ll ” denotes much worse than. Intuitively, the best case is $g(x) = \pi(x)$.

Since we need that

$$\lim_{n \rightarrow \infty} (\hat{C} - C) = 0,$$

and all three scenarios satisfy it, the only difference lies in the need of data, or the convergence rate, or the variance,

$$\lim_{n \rightarrow \infty} \|\hat{C} - C\|^2 = 0.$$

The approximation distribution $g(x)$ serves the purpose of a lever that can be used to handle $\pi(x)$. The Greek mathematician Archimedes (212 BC) is known for his famous remark:

“Give me a place to stand and with a lever I shall move the earth”.

Inspired by him, we could refer to the approximation distribution $g(x)$ as an *Archimedes lever* for $\pi(x)$.

Example 2.1. For the third case above, here is an example of an Archimedes lever.

$$\pi(x) = \frac{1}{Z} \exp \left\{ - \sum_{i=1}^K \beta_i h_i(x) \right\}, \quad g(x) = \frac{1}{Z'} \exp \left\{ - \sum_{i=1}^{K' < K} \beta_i h_i(x) \right\}$$

Example 2.2. In the Gaussian case, we can use the following scheme

$$\pi(x) = \frac{1}{Z} \exp \{ -(ax^2 + bx + c) \}, \quad g(x) = \frac{1}{Z'} \exp \{ -ax^2 \}$$

Usually, we use a set of weighted samples, $\{(x_i, \omega_i), i = 1, 2, \dots, m\}$, which is called “empirical distribution” to represent $\pi(x)$.

When $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ is high-dimensional, we can use two ways to simplify.

$$\begin{aligned} g(\mathbf{x}) &= g(x_1, x_2, \dots, x_n) \\ &\cong g(x_1) \cdot g(x_2) \cdot \dots \cdot g(x_n) \quad (\text{by factorization}) \end{aligned} \tag{2.4}$$

$$\cong g(x_1) \cdot g(x_2) \cdot g(x_3|x_2) \cdot g(x_4|x_1, x_2) \cdot \dots \quad (\text{by factorization}) \tag{2.5}$$

In (2.4), we are assuming that each x_i are independent, and we sample each dimension independently, however, the truth is they are always dependent. So we need to use (2.5) to simplify the problem.

Since $\hat{C} = \frac{1}{m} \sum_{i=1}^m w(x_i) h(x_i)$, $\text{var}_m(\hat{C}) = \frac{1}{m} \text{var}_1(\hat{C})$. This shows that when we have sufficiently many samples, the total variance will go to zero, and the rate of convergence is $\frac{1}{m}$, regardless of the dimension n ! The three plots in Figure 2.4 illustrate the story. The convergence of the left figure is quick while the convergence rate of the middle one can be slow. The right plot might have problems with weights blowing up to ∞ .

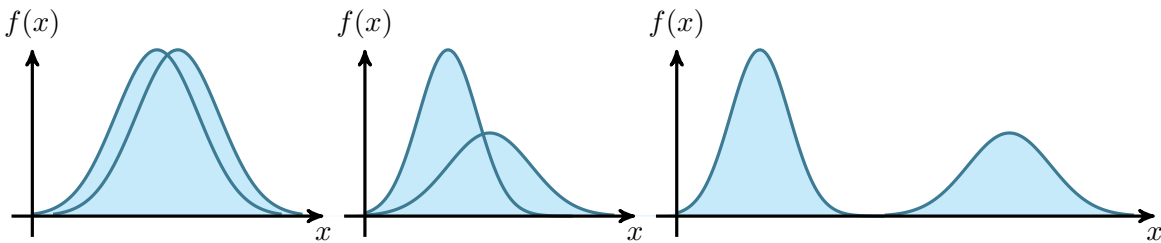


Figure 2.4: Left: when $g(x)$ and $\pi(x)$ are close, convergence is fast. Middle: when $g(x)$ and $\pi(x)$ are far, convergence is much slower. Right: In general, $g(x)$ should be nonzero wherever $\pi(x)$ is nonzero, which might not happen in this case.

The heuristics for measuring the effectiveness of samples from $g(x)$ is to measure the variance of the “weights”. A useful “rule of thumb” is to use the *effective sample size* (ESS) to measure how different the trial distribution is from the target distribution. Suppose m independent samples are generated from $g(x)$; then, the ESS is defined as

$$\text{ESS}(m) = \frac{m}{1 + \text{var}[\omega(x)]} \tag{2.6}$$

In the ideal case of $g(x) = \pi(x)$, then $\omega(x) = 1$, $\text{var}_g[\omega(x)] = 0$, thus the whole samples are effective.

Since the target distribution π is known only up to a normalizing constant in many problems, the variance of the *normalized* weight needs to be estimated by the *coefficient of variation* of the unnormalized weight:

$$\text{cv}^2(\omega) = \frac{1}{m-1} \sum_{i=1}^m \frac{(\omega_i - \bar{\omega})^2}{\bar{\omega}^2} \quad (2.7)$$

Generalization: *Stratified sampling* – a method to reduce $\text{var}(\hat{C})$. Suppose the space Ω is the union of a number of disjoint subspaces $\Omega = \cup_{j=1}^M \Omega_j$. In each subspace, Ω_j , we can define different $g_j(x)$ as trial distributions. Thus, we will have

$$C = \sum_{j=1}^M \int_{\Omega_j} g_j(x) \cdot \frac{\pi(x)}{g_j(x)} \cdot h(x) dx \quad (2.8)$$

In the computation process, we can ignore the overlap of $g_i(x)$'s in higher dimensional space.

2.3 Sequential Importance Sampling(SIS)

In high-dimensional space, it is usually very hard to find an effective $g(x)$. Suppose we can decompose \mathbf{x} as $\mathbf{x} = (x_1, \dots, x_n)$ by chain rule. Then, our trial density can be constructed as

$$g(\mathbf{x}) = g_1(x_1) \cdot g_2(x_2|x_1) \cdots g_n(x_n|x_1, \dots, x_{n-1}). \quad (2.9)$$

Usually this is impractical, but in some cases it can be done if the $\pi(x)$ is similarly factorized. Corresponding to the decomposition of \mathbf{x} , we can rewrite the target density as

$$\pi(\mathbf{x}) = \pi_1(x_1) \cdot \pi_2(x_2|x_1) \cdots \pi_n(x_n|x_1, \dots, x_{n-1}). \quad (2.10)$$

and the importance weight as

$$\omega(\mathbf{x}) = \frac{g(\mathbf{x})}{\pi(\mathbf{x})} = \frac{g_1(x_1) \cdot g_2(x_2|x_1) \cdots g_n(x_n|x_1, \dots, x_{n-1})}{\pi_1(x_1) \cdot \pi_2(x_2|x_1) \cdots \pi_n(x_n|x_1, \dots, x_{n-1})} \quad (2.11)$$

In the following, we will discuss two examples.

- 1) Self-Avoid-Walk – growing polymer problem;
- 2) Non-linear/particle filtering for tracking.

2.3.1 Application: the number of self-avoiding walks

The self-avoiding random walk (SAW) in a two- or three-dimensional lattice space is the problem of finding how many self-avoiding walks exist in a given domain.

We can use a “hard-core” model to describe it. A chain of atoms $\mathbf{x} = (x_1, x_2, \dots, x_N)$ is connected by bonds (covalent). For clarity, we assume each molecule to be a point in 2D/3D space/lattice, and a bond has length = 1, so the potential is “hard-core”. In the 2D or 3D space, the chain is not allowed to intersect itself.

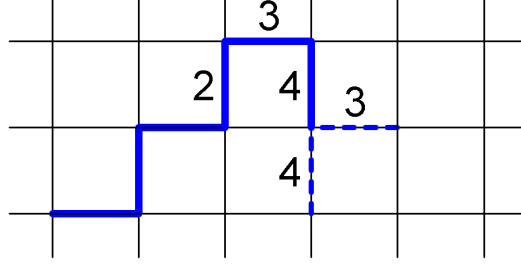


Figure 2.5: The self avoiding walk.

In this section we will focus on the 2D domain $\{0, 1, \dots, n\} \times \{0, 1, \dots, n\}$. Suppose we always start from position $(0, 0)$, i.e. lower-left corner, as illustrated in Figure 2.5.

Representing the moves to left/right/up/down by the numbers 1,2,3,4 respectively, the Gibbs/Boltzmann distribution for the chain of a SAW is

$$\pi(x) = \text{unif}[\Omega], \quad \Omega = \{x : \text{SAW}(x) = 1\}, \quad x \in \{1, 2, 3, 4\}^n$$

We are interested in the total number of SAWs. To solve this problem, we use Monte Carlo integration. We design a trial probability $g(x)$ for a SAW x that is easier to sample. Then we sample a number M of SAWs from $g(x)$, and we estimate the total count by

$$||\Omega|| = \theta = \sum_{x \in \Omega} 1 = \sum_{x \in \Omega} \frac{1}{g(x)} g(x) \cong \frac{1}{M} \sum_{i=1}^M \frac{1}{g(x_i)} \quad (2.12)$$

where $\frac{1}{g(x_i)}$ serves as the weight $\omega(x_i)$ of x_i .

The trial probability $g(x)$ covers all possible paths, so we can use it to compute the size of many subsets of the set of SAWs, such as the set of all SAW that start in a corner and end in another corner, or the set of SAWs of length n . We don't need to worry about the normalization constant under this new subset.

Therefore, the problem lies in how to design $g(x)$ and there are several ways to do that. We try three different types of $g(x)$ in a 2D grid with $n = 10$ to generate $M = 10^7$ samples, respectively.

a) Design 1. As an initial method, we use

$$g_1(x) = \prod_{j=1}^m \frac{1}{k_j}$$

where m is the total length of the path, and k_j is the number of possible choices at j -th step. With $M = 10^7$ samples, the estimated number of SAWs is $K_1 = 3.3844 \cdot 10^{25}$. The length distribution of sampled walks and the longest walk is visualized in Figure 2.6. Since we do not constrain the length of the walk, the distribution obtained resembles a Gaussian shape.

b) Design 2. As an alternative design of the trial distribution, we introduce an early termination rate $\epsilon = 0.1$ at each step. So

$$g_2(x) = (1 - \epsilon)^m \prod_{j=1}^m \frac{1}{k_j}$$

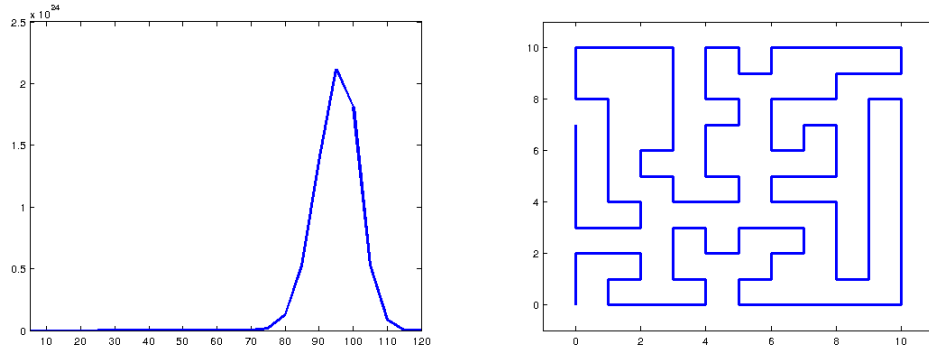


Figure 2.6: Left: distribution of SAW lengths for design 1. Right: longest SAW (length 114).

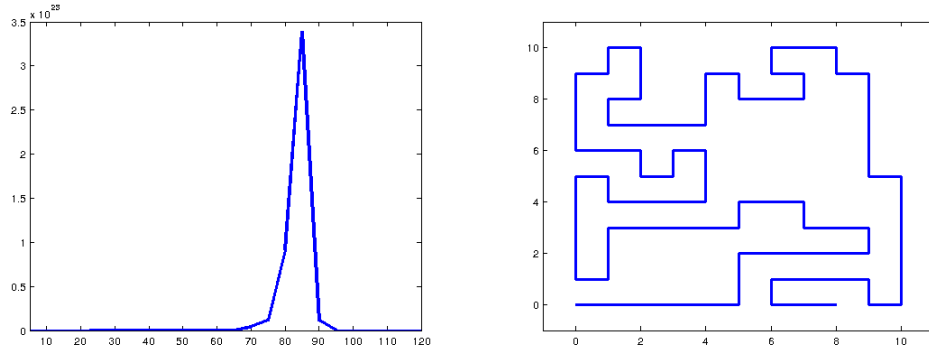


Figure 2.7: Left: distribution of SAW lengths for design 2. Right: longest SAW (length 87).

Clearly, in this case we are expected to get shorter walks than in previous situation. The length distribution of sampled walks and the longest walk are shown in Figure 2.7. The estimated number of SAWs is $K_2 = 6.3852 \cdot 10^{25}$.

- c) Design 3. As the third design, we want to favor long walks. For any walk that longer than 50, we generate $u = 5$ more children based on it and reweigh each of the children by $w_0 = w/u$. The length distribution of sampled walks and the longest walk are shown in Figure 2.8. The estimated number of SAWs is $K_3 = 7.3327 \cdot 10^{25}$.

The log-log plot for the estimated number of SAWs K against the sample size m is shown in Figure 2.9. It is obvious that the design 3 converges the fastest and design 2 converges the slowest among the three.

Other ways to design the trial probability $g(x)$ are:

- Stop at any time (see design 2)
- Fix the length N
- Enrich. Encourage longer samples – begin with a certain length (see design 3)
- Global guidance? $(0, 0) \rightarrow (n, n)$
- Other heuristics to define the trial probability $g(x)$?

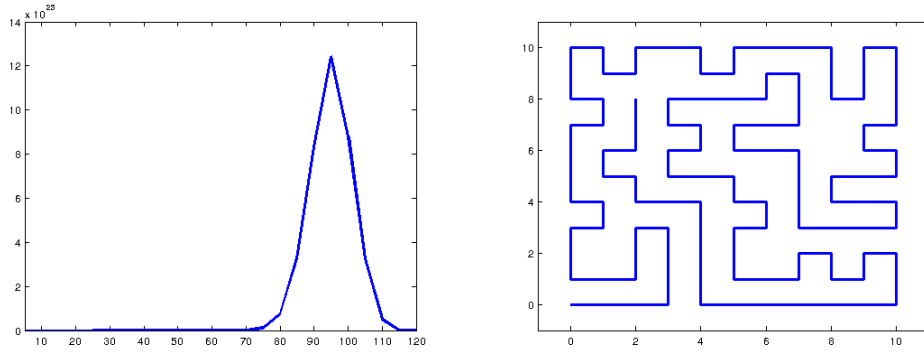


Figure 2.8: Left: distribution of SAW lengths for design 3. Right: longest SAW (length 115).

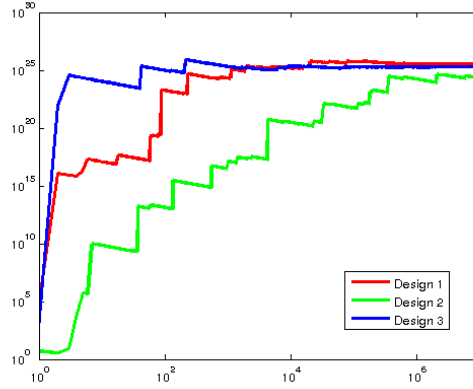


Figure 2.9: Convergence rate comparison of the three designs.

SAW from $(0, 0)$ to (n, n) . To get samples that reach (n, n) , for those samples that don't reach the target, we resample until a desired one is obtained and then reweigh it by $w_0 = w/u$, where u is number of attempts. This means the more attempts we tried, the less weight this sample get. Generating 10^6 samples, we estimate the total number of SAWs from $(0, 0)$ to (n, n) to be about $1.7403 \cdot 10^{24}$ (which is very close to the true value $1.5687 \cdot 10^{24}$).

2.3.2 Application: particle filtering for tracking objects in video

Assume we have an object tracking problem where the state at time t is denoted by \mathbf{x}_t and the observed image features are denoted by \mathbf{z}_t . Denote the state history by $\mathcal{X}_t = \{\mathbf{x}_1, \dots, \mathbf{x}_t\}$ and the feature history by $\mathcal{Z}_t = \{\mathbf{z}_1, \dots, \mathbf{z}_t\}$.

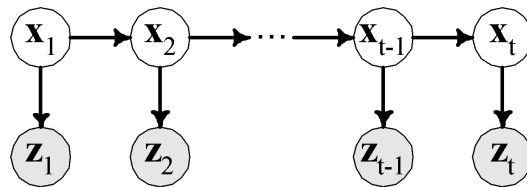


Figure 2.10: The observations \mathbf{z}_t are independent with each other and with respect to the dynamical process \mathbf{x}_t . The dynamical process \mathbf{x}_t has a Markov dependency on only the previous state \mathbf{x}_{t-1} .

We assume that the object dynamics follows a Markov chain, i.e. the current state depends only on the preceding state, independent of the state history, as illustrated in Figure 2.10.

$$p(\mathbf{x}_{t+1}|\mathcal{X}_t) = p(\mathbf{x}_{t+1}|\mathbf{x}_t)$$

The observations \mathbf{z}_t are assumed independent with each other and with respect to the dynamical process, as illustrated in Figure 2.10. We need to estimate $p(\mathbf{x}_{t+1}|\mathcal{Z}_{t+1})$, i.e. the distribution of the state x_{t+1} given all the data received so far. We have:

$$\begin{aligned} p(\mathbf{x}_{t+1}|\mathcal{Z}_{t+1}) &= p(\mathbf{x}_{t+1}|\mathbf{z}_{t+1}, \mathcal{Z}_t) = \frac{p(\mathbf{x}_{t+1}, \mathbf{z}_{t+1}|\mathcal{Z}_t)}{p(\mathbf{z}_{t+1}|\mathcal{Z}_t)} \\ &\propto p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})p(\mathbf{x}_{t+1}|\mathcal{Z}_t) = p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t) \end{aligned}$$

because \mathbf{z}_{t+1} is independent of \mathcal{Z}_t . We can compute

$$p(\mathbf{x}_{t+1}|\mathcal{Z}_t) = \int p(\mathbf{x}_{t+1}, \mathbf{x}_t|\mathcal{Z}_t)d\mathbf{x}_t = \int p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t)d\mathbf{x}_t$$

so we get

$$p(\mathbf{x}_{t+1}|\mathcal{Z}_{t+1}) \propto \int p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t)d\mathbf{x}_t$$

The probability $p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})$ can be considered the bottom-up probability of detection, while the product $p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t)$ is the prediction based on the dynamic model.

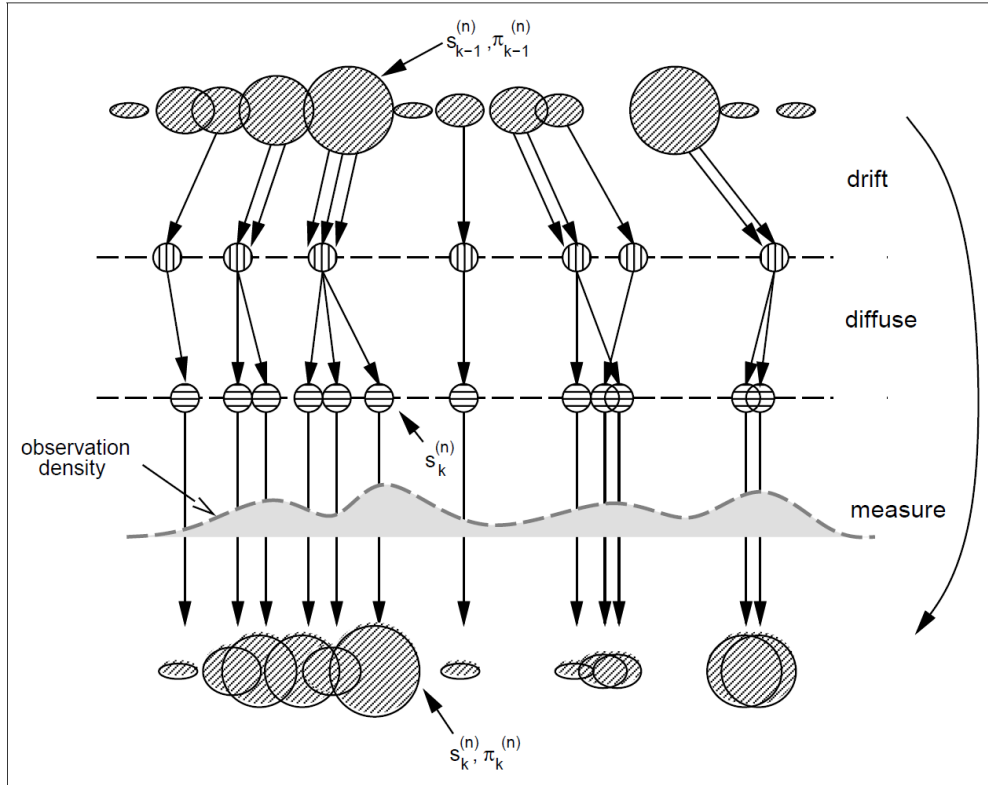


Figure 2.11: One time step of the CONDENSATION algorithm [96].

The CONDENSATION algorithm [96] represents the distribution of $p(\mathbf{x}_t|\mathcal{Z}_t)$ using importance sampling as a weighted sample set $\{\mathbf{s}_t^{(n)}, n = 1, \dots, N\}$ with weights $\pi_t^{(n)}$. One step of the algorithm is illustrated in Figure 2.11 and described in Figure 2.12.

```

Input: Sample set  $\{(\mathbf{s}_{t-1}^{(n)}, \pi_{t-1}^{(n)}), n = 1, \dots, N\}$ 
Compute the cumulative distribution values  $c_{t-1}^{(k)} = \sum_{i=1}^k \pi_{t-1}^{(i)}$ .
for  $n = 1, \dots, N$  do
  Drift: sample  $\mathbf{s}_t^{(n)}$  from the cumulative distribution  $c_{t-1}^{(k)}, k = 1, \dots, N$ .
  Diffuse: Sample  $\mathbf{s}_t^{(n)}$  from the dynamical model  $\mathbf{s}_t^{(n)} \sim p(\mathbf{x}_t | \mathbf{x}_{t-1} = \mathbf{s}_t^{(n)})$ .
  Measure and weight the sample  $\mathbf{s}_t^{(n)}$  as  $\pi_t^{(n)} = p(\mathbf{z}_t | \mathbf{x}_t = \mathbf{s}_t^{(n)})$ .
end for
Normalize  $\pi_t^{(n)}$  such that  $\sum_{n=1}^N \pi_t^{(n)} = 1$ .
Output: Sample set  $\{(\mathbf{s}_t^{(n)}, \pi_t^{(n)}), n = 1, \dots, N\}$ 

```

Figure 2.12: One time step of the CONDENSATION algorithm [96].

Application: Curve Tracking

The curve at time t is represented as a curve $\mathbf{r}(s, t)$ parameterized as a B-spline:

$$\mathbf{r}(s, t) = (B(s)Q^x(t), B(s)Q^y(t)), s \in [0, L]$$

where $B(s) = (B_1(s), \dots, B_{N_B}(s))^T$ is a vector of B-spline basis functions. The vector $X_t = (Q^x, Q^y)^T$ contains the coordinates of the spline control points.

The dynamical model is a first order autoregressive model:

$$\mathbf{x}_t - \bar{\mathbf{x}} = A(\mathbf{x}_{t-1} - \bar{\mathbf{x}}) + B\mathbf{w}_t$$

where \mathbf{w}_t are independent vectors of i.i.d $\mathcal{N}(0, 1)$ and $\mathbf{x}_t = \begin{pmatrix} X_{t-1} \\ X_t \end{pmatrix}$.

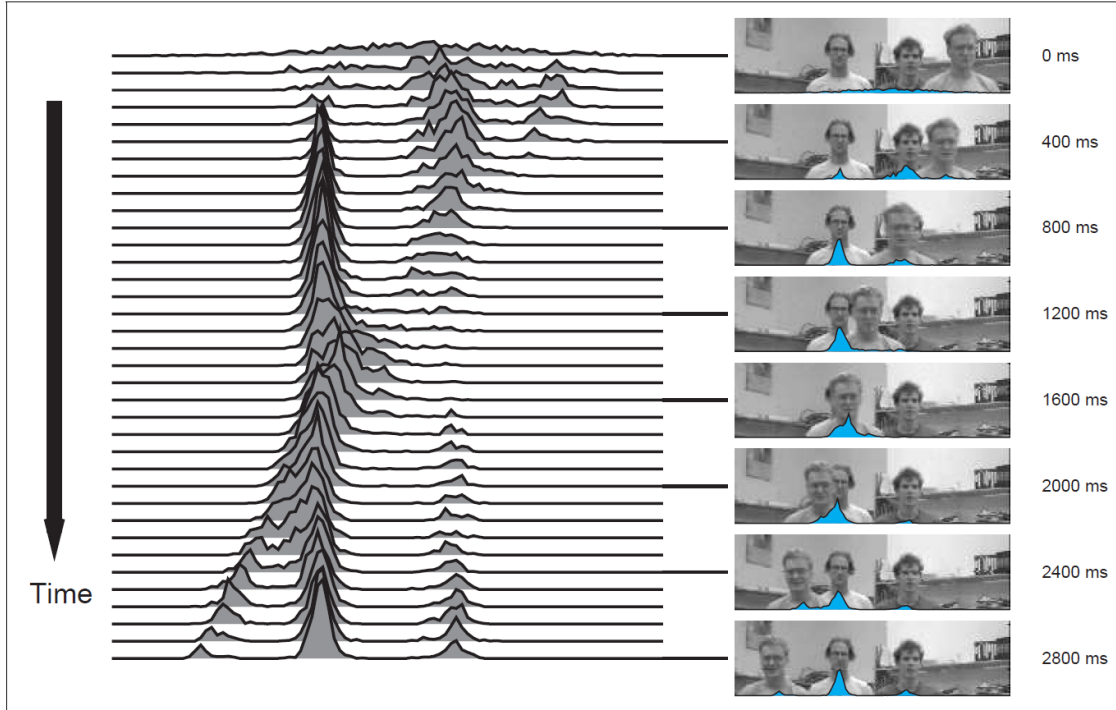


Figure 2.13: 1D projection of the state density across multiple frames of a video, from [96]. The dynamical model can also be expressed as:

$$p(\mathbf{x}_t | \mathbf{x}_{t-1}) \propto \exp\left(-\frac{1}{2} \|(\mathbf{x}_t - \bar{\mathbf{x}}) - A(\mathbf{x}_{t-1} - \bar{\mathbf{x}})\|^2\right)$$

The observation model for 2D curves can be for example:

$$p(\mathbf{z}|\mathbf{x}) \propto \exp \left(- \sum_{m=1}^M \frac{1}{2rM} f(\mathbf{z}_i(\frac{m}{M}) - \mathbf{r}(\frac{m}{M}); \mu) \right)$$

where r, μ are constants, M is the number of points for curve discretization, $f(x; \mu) = \min(x^2, \mu^2)$, and $\mathbf{z}_1(s)$ is the closest image feature to $\mathbf{r}(s)$:

$$\mathbf{z}_i(s) = \mathbf{z}(s') \text{ where } s' = \underset{s' \in g^{-1}(s)}{\operatorname{argmin}} |\mathbf{r}(s) - \mathbf{z}(s')|$$

An example of a tracking result obtained using this model is shown in Figure 2.13.

2.3.3 Summary of the SMC Framework

In Sequential Monte Carlo, the term "sequential" has two meanings:

1. Unfold a joint state $\mathbf{x} = (x_1, x_2, \dots, x_d)$ by component, as in the Self-Avoiding Walks from Section 2.3.1.
2. Update X_t in time as in the particle filter from Section 2.3.2.

The following issues arise in the design of SMC/SIS:

1. Choice of the trial probability. For example, in particle filtering

$$p(\mathbf{x}_{t+1}|\mathcal{Z}_{t+1}) = \int_{\mathbf{x}_t} p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t)d\mathbf{x}_t$$

one may generate particles using

- a) A data-driven approach by sampling from $p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})$ (tracking by detection). This is important when the target is lost.
- b) A dynamics-driven approach by sampling from $p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathcal{Z}_t)$ and reweighing by the evidence $p(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})$.

An even better alternative is to generate particles using both the data-driven and the dynamics driven approaches as different channels, which can complete each other depending on the data quality at each time step.

2. How to rejuvenate the sample – punning, enriching, re-sampling/reweighing.

Example 2.3. In the Self Avoiding Walk, suppose we have a partial sample $x^{(j)}$ of length n , with n large enough and with trial probability $g_1(x^{(j)}) = \frac{1}{k_1} \frac{1}{k_2} \dots \frac{1}{k_n}$, which is very small, contributing $w^{(j)} = k_1 \dots k_n$ to the final summation, which is very large. One idea is to make k copies of $x^{(j)}$, each copy with weight $\frac{1}{k} w^{(j)}$. This is equivalent to changing the proposal probability $g(x)$ so that $g(x^{(j)})$ is k times larger.

Example 2.4. Similarly in particle filtering, we resampled the weighted sample set $\hat{S} = \{(\hat{x}^{(j)}, w^{(j)}), j = 1, \dots, m\}$ with an equal-weight set $S = \{x^{(j)}, j = 1, \dots, m\}$ containing repeated copies of the strong samples, so that in the next step, the strong samples have many offsprings.

In both cases, the idea significantly improves performance.

Criterion for resampling. In SMC we can monitor the sample $S = \{x^{(j)}, w^{(j)}, j = 1, \dots, m\}$ by the $\text{var}(\mathbf{w})$ of the weight vector $\mathbf{w} = (w^{(1)}, \dots, w^{(m)})$, or the coefficient of variation

$$CV(\mathbf{w}) = \sqrt{\frac{\sum_{j=1}^m (w^{(j)} - \bar{w})^2}{(m-1)\bar{w}^2}}$$

When $CV(w)$ is too large, $CV(\mathbf{w}) > c_0$, a resampling step is necessary.

Reweighting. When resampling $S = \{x^{(j)}, w^{(j)}, j = 1, \dots, m\}$, we may not always have to use the weight vector $\mathbf{w} = (w^{(1)}, \dots, w^{(m)})$ to generate weights proportionally. Instead, we can chose an arbitrary vectors with nonzero entries $\mathbf{a} = (a^{(1)}, \dots, a^{(m)})$, $a_i > 0$ and reweigh the samples as $w^{*(j)} = w^{(j)}/a^{(j)}$. The weights \mathbf{a} should be designed to penalize redundant samples and to encourage distinctive samples.

2.4 Application: Ray Tracing by SMC

Another application of SMC is in ray tracing [197] for calculating the radiance of a surface given the description of the light sources.

Given the incident radiance function $L_i(\mathbf{x}, \omega_i)$ at point \mathbf{x} , the reflected radiance follows the *reflectance equation*:

$$L_r(\mathbf{x}, \omega_r) = \int_{S^2} f_r(\mathbf{x}, \omega_i \leftrightarrow \omega_r) L_i(\mathbf{x}, \omega_i) |\cos(\theta_i)| d\sigma(\omega_i) \quad (2.13)$$

where f_r is the bidirectional reflectance distribution function (BRDF), S^2 is the unit sphere in 3D, σ is the solid angle measure, θ_i is the angle between ω_i and the surface normal at \mathbf{x} .

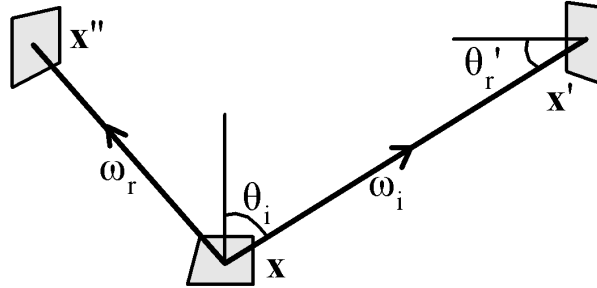


Figure 2.14: Illustration of the reflectance equation.

If we want to use only points in the scene, we could also write the reflectance equation as follows:

$$L_r(\mathbf{x} \rightarrow \mathbf{x}'') = \int_{\mathcal{M}} f_r(\mathbf{x}' \leftrightarrow \mathbf{x} \leftrightarrow \mathbf{x}'') L_i(\mathbf{x}' \rightarrow \mathbf{x}) G(\mathbf{x} \leftrightarrow \mathbf{x}') dA(\mathbf{x}') \quad (2.14)$$

where $G(\mathbf{x} \leftrightarrow \mathbf{x}') = V(\mathbf{x} \leftrightarrow \mathbf{x}') \frac{\cos(\theta'_r) \cos(\theta_i)}{\|\mathbf{x} - \mathbf{x}'\|^2}$ and A is the measure of surface area, θ'_r and θ_i are the angles between $\mathbf{x} \leftrightarrow \mathbf{x}'$ and the surface normals at \mathbf{x} and \mathbf{x}' . The function $V(\mathbf{x} \leftrightarrow \mathbf{x}')$ is 1 if \mathbf{x} and \mathbf{x}' are mutually visible and 0 else.

We arrive at the global illumination problem of finding the equilibrium radiance distribution L that satisfies:

$$L(\mathbf{x} \rightarrow \mathbf{x}'') = L_e(\mathbf{x} \rightarrow \mathbf{x}'') + \int_{\mathcal{M}} f_r(\mathbf{x}' \leftrightarrow \mathbf{x} \leftrightarrow \mathbf{x}'') L(\mathbf{x}' \rightarrow \mathbf{x}) G(\mathbf{x} \leftrightarrow \mathbf{x}') dA(\mathbf{x}')$$

where the emitted radiance distribution L_e is given. This is the three point rendering equation [98]. It can be written concisely as $L = L_e + \mathcal{T}L$ where \mathcal{T} is the light transport operator. Under weak assumptions, the solution is the Neumann series:

$$L = \sum_{i=1}^{\infty} \mathcal{T}^i L_e$$

2.4.1 Example: glossy highlights

Consider the ray tracing problem of rendering the highlight produced by an area light source S on a nearby glossy surface, as illustrated in Figure 2.15. There are two obvious strategies to use MC to approximate the reflected radiance, using eq. (2.13) and (2.14) respectively. For both strategies we use importance sampling, where the samples are x_1, \dots, x_n obtained from a distribution $p(x)$. Then the integral is approximated as:

$$\int_{\Omega} f(x) d\mu(x) \approx \frac{1}{n} \sum_{i=1}^n \frac{f(x_i)}{p(x_i)}$$

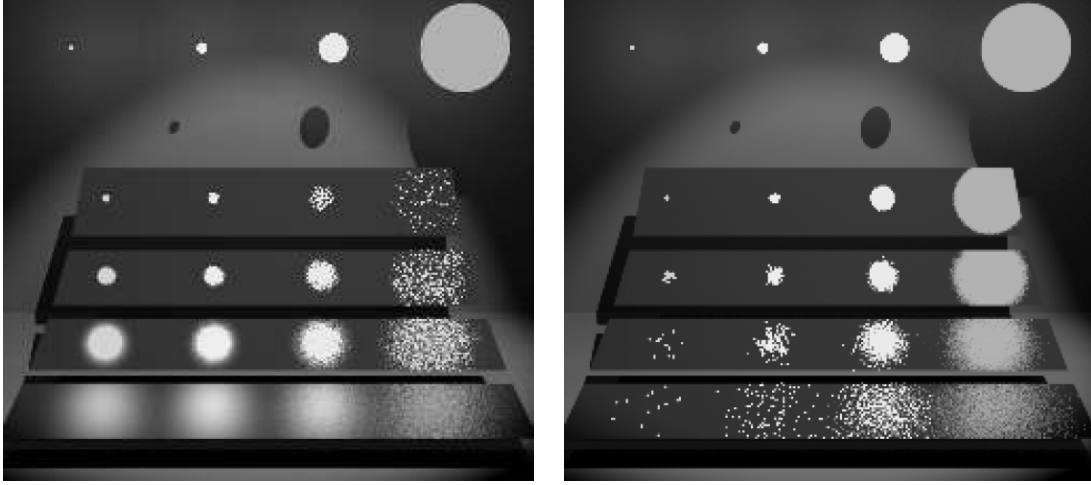
With *area sampling* we randomly choose points on the surface to approximate (2.14). The points could for example be chosen uniformly on S with respect to the surface area or the emitted power.

With *directional sampling* we approximate (2.13) using random samples of the direction ω_i . usually $p(\omega_i) d\sigma(\omega_i)$ is chosen proportional to $f_r(\mathbf{x}\omega_i \leftrightarrow \omega_r)$ or to $f_r(\mathbf{x}\omega_i \leftrightarrow \omega_r) |\cos(\theta_i)|$.

In Figure 2.15 are shown examples of renderings using different sampling methods. The scene contains four spherical light sources of different radii and color, and a spotlight overhead. All spherical light sources emit the same total power. There are also four shiny rectangular plates of varying surface roughness, tilted so that the reflected light sources are visible. Given a viewing ray that strikes a glossy surface, images (a), (b), (c) use different techniques for the highlight calculation. All images are 500 by 450 pixels. The MC techniques are:

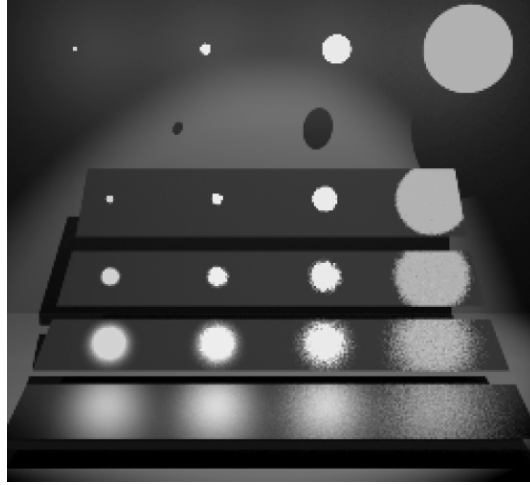
- (a) Area sampling. A sample direction ω_i is chosen uniformly (with respect to solid angle) within the cone of directions of each light source, using 4 samples per pixel.
- (b) Directional sampling. The direction ω_i is chosen with probability proportional to the BRDF $f_r(\mathbf{x}\omega_i \leftrightarrow \omega_r) d\sigma(\omega_i)$, with 4 samples per pixel.
- (c) A weighted combination of the samples from (a) and (b) is computed, using the power heuristic with $\beta = 2$.

The glossy BRDF is a symmetric, energy-conserving variation of the Phong model. The Phong exponent is $n = 1/r - 1$, where $r \in (0, 1)$ is a surface roughness parameter. The glossy surfaces also have a small diffuse component.



a) Area sampling.

b) Directional sampling.



c) Weighted combination of the samples from (a) and (b)

Figure 2.15: Sampling of glossy highlights from area light sources [197].

2.5 Preserving Sample Diversity in Importance Sampling

For simplicity of notation, we denote by $p(\mathbf{x})$ an arbitrary distribution in space Ω . For segmentation problems with Bayesian Inference, we observe that $p(\mathbf{x})$ has two important properties.

1. $p(\mathbf{x})$ has an enormous number of local maxima (called modes in statistics). A significant mode corresponds to a distinct interpretation of the image, and the cloud surrounding a mode is local small perturbation of the region boundaries or model parameters. These significant modes of $p(\mathbf{x})$, denoted by $\mathbf{x}_i, i = 1, 2, \dots$, are well separated from each other due to the high dimensions.
2. Each mode \mathbf{x}_i has a weight $\omega_i = p(\mathbf{x}_i)$, and its energy is defined as $E(\mathbf{x}_i) = -\log p(\mathbf{x}_i)$. The energies of these modes are uniformly distributed in a broad range $[E_{\min}, E_{\max}]$, say, $[1000, 10,000]$. For example, it is normal to have solutions (or local maxima) whose energies differ in the order of 500 or more. Thus their probability (weights) differ in the order of

\exp^{-500} , and our perception is interested in those “trivial” local modes!

Intuitively, it helps to imagine that $p(\mathbf{x})$ in Ω is distributed like the mass of the universe. Each star is a mode as local maximum of the mass density. The significant and developed stars are well separated apart from each other and their masses could differ in many orders of magnitudes. The above metaphor leads us to a mixture of Gaussian representation of $p(\mathbf{x})$. For a large enough N , we have,

$$p(\mathbf{x}) = \frac{1}{\omega} \sum_{j=1}^N \omega_j G(\mathbf{x} - \mathbf{x}_j, \sigma_j^2), \quad \omega = \sum_{j=1}^N \omega_j.$$

We denote by

$$S_o = \{(\omega_j, \mathbf{x}_j), j = 1, 2, \dots, N\}, \quad \sum_{j=1}^N \omega_j = \omega,$$

the set of weighted particles (or modes). Thus our task is to select $K \ll N$ particles S from S_o . We define a mapping from the index in S to the index in S_o ,

$$\tau : \{1, 2, \dots, K\} \longrightarrow \{1, 2, \dots, N\}.$$

Therefore,

$$S = \{(\omega_{\tau(i)}, \mathbf{x}_{\tau(i)}); i = 1, 2, \dots, K\}$$

S encodes a non-parametric model for approximating $p(\mathbf{x})$ by

$$\hat{p}(\mathbf{x}) = \frac{1}{\alpha} \sum_{i=1}^K \omega_{\tau(i)} G(\mathbf{x} - \mathbf{x}_{\tau(i)}, \sigma_{\tau(i)}^2), \quad \alpha = \sum_{i=1}^K \omega_{\tau(i)}.$$

Our goal is to compute

$$S^* = \arg \min_{|S|=K} D(p||\hat{p}).$$

For notational simplicity, we assume all Gaussians have the same variance in approximating $p(\mathbf{x})$, $\sigma_j = \sigma, j = 1, 2, \dots, N$. By analogy, all “stars” have the same volume, but differ in weight. With the two properties of $p(\mathbf{x})$, we can approximately compute $D(p||\hat{p})$ in the following. We start with an observation for the KL-divergence for Gaussian distributions.

Let $p_1(x) = G(\mathbf{x} - \mu_1; \sigma^2)$ and $p_2(x) = G(\mathbf{x} - \mu_2; \sigma^2)$ be two Gaussian distributions, then it is easy to check that

$$D(p_1||p_2) = \frac{(\mu_1 - \mu_2)^2}{2\sigma^2}.$$

We partition the solution space Ω into disjoint domains

$$\Omega = \cup_{i=1}^N D_i, \quad D_i \cap D_j = \emptyset, \quad \forall i \neq j.$$

D_i is the domain where $p(\mathbf{x})$ is decided by a particle (ω_i, \mathbf{x}_i) . The reason for this partition is that the particles in S are far apart from each other in high dimensional space and their energy varies significantly as the two properties state. Within each domain D_i , it is reasonable to assume that $p(\mathbf{x})$ is dominated by one term in the mixture and the other $N - 1$ terms are neglectable.

$$p(\mathbf{x}) \approx \frac{\omega_i}{\omega} G(\mathbf{x} - \mathbf{x}_i; \sigma^2), \quad \mathbf{x} \in D_i, \quad i = 1, 2, \dots, N.$$

The size of D_i is much larger than σ^2 . After removing $N - K$ particles in the space, a domain D_i is dominated by a nearby particle that is selected in S .

We define a second mapping function

$$c : \{1, 2, \dots, N\} \rightarrow \{1, 2, \dots, K\},$$

so that $\hat{p}(\mathbf{x})$ in D_i is dominated by a particle $\mathbf{x}_{\tau(c(i))} \in S_K$,

$$\hat{p}(\mathbf{x}) \approx \frac{\omega_{c(i)}}{\alpha} G(\mathbf{x} - \mathbf{x}_{\tau(c(i))}; \sigma^2), \quad \mathbf{x} \in D_i, i = 1, 2, \dots, N.$$

Intuitively, the N domains are partitioned into K groups, each of which is dominated by one particle in S_K . Thus we can approximate $D(p||\hat{p})$.

$$\begin{aligned} D(p||\hat{p}) &= \sum_{n=1}^N \int_{D_n} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{\hat{p}(\mathbf{x})} d\mathbf{x} \\ &= \sum_{n=1}^N \int_{D_n} \frac{1}{\omega} \sum_{i=1}^N \omega_i G(\mathbf{x} - \mathbf{x}_i; \sigma^2) \log \frac{\frac{1}{\omega} \sum_{i=1}^N \omega_i G(\mathbf{x} - \mathbf{x}_i; \sigma^2)}{\frac{1}{\alpha} \sum_{j=1}^K \omega_{\tau(j)} G(\mathbf{x} - \mu_{\tau(j)}; \sigma^2)} d\mathbf{x} \\ &\approx \sum_{n=1}^N \int_{D_n} \frac{\omega_n}{\omega} G(\mathbf{x} - \mathbf{x}_n; \sigma^2) \left[\log \frac{\alpha}{\omega} + \log \frac{\omega_n G(\mathbf{x} - \mathbf{x}_n; \sigma^2)}{\omega_{\tau(c(n))} G(\mathbf{x} - \mathbf{x}_{\tau(c(n))}; \sigma^2)} \right] d\mathbf{x} \\ &= \sum_{n=1}^N \frac{\omega_n}{\omega} \left[\log \frac{\alpha}{\omega} + \log \frac{\omega_n}{\omega_{\tau(c(n))}} + \frac{(\mathbf{x}_n - \mathbf{x}_{\tau(c(n))})^2}{2\sigma^2} \right] \\ &= \log \frac{\alpha}{\omega} + \sum_{n=1}^N \frac{\omega_n}{\omega} \left[(E(\mathbf{x}_{\tau(c(n))}) - E(\mathbf{x}_n)) + \frac{(\mathbf{x}_n - \mathbf{x}_{\tau(c(n))})^2}{2\sigma^2} \right] = \hat{D}(p||\hat{p}). \end{aligned} \tag{2.15}$$

Equation (2.15) has some intuitive meanings. The second term suggests that each selected $\mathbf{x}_{\tau(c(i))}$ should have large weight $\omega_{\tau(c(i))}$. The third term is the attraction forces from particles in S_o to particles in S . Thus it helps to pull apart the particles in S_k and also plays the role of encouraging to choose particles with big weight like the second term.

To demonstrate the goodness of approximation $\hat{D}(p||\hat{p})$ to $D(p||\hat{p})$, we show two experiments below.

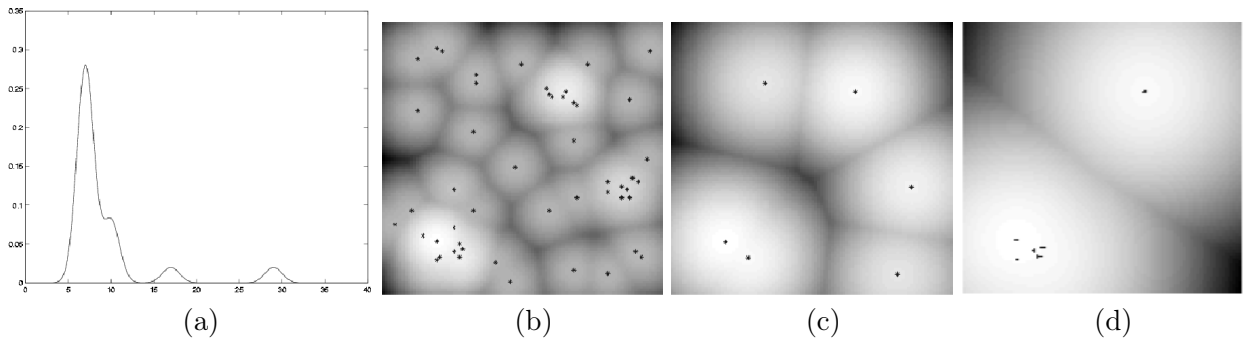


Figure 2.16: (a) A 1D dist. $p(\mathbf{x})$ with four particles $\mathbf{x}_i, i = 1, 2, 3, 4$. (b) A 2D dist $p(\mathbf{x})$ with 50 particles, we show $\log p(\mathbf{x})$ in the image for visualization. (c) $\hat{p}_1(x)$ with 6 particles that minimizes $D(p||\hat{p})$ or $D(p||\hat{p})$. (d) $\hat{p}_2(x)$ with 6 particles that minimizes $|p - \hat{p}|$.

Figure 2.16.a displays a 1D distribution $p(\mathbf{x})$ which is a mixture of $N = 4$ Gaussians (particles). We index the centers from left to right $\mathbf{x}_1 < \mathbf{x}_2 < \mathbf{x}_3 < \mathbf{x}_4$. Suppose we want to choose $K = 3$ particles for S_K and $\hat{p}(\mathbf{x})$. Table 2.1 displays the distances between $p(\mathbf{x})$ and $\hat{p}(\mathbf{x})$ over the four possible combinations. The second row shows the KL-divergence $D(p||\hat{p})$ and the third row is $\hat{D}(p||\hat{p})$. The approximation is very accurate given the particles are well separable.

Both measures choose (x_1, x_3, x_4) as the best S . Particle x_2 is not favored by the KL-divergence because it is near x_1 , although it has much higher weight than x_3 and x_4 . The fourth row shows the absolute value of the difference between $p(x)$ and $\hat{p}(x)$. This distance favors (x_1, x_2, x_3) and (x_1, x_2, x_4) . In comparison, the KL-divergence favors particles that are apart from each other and picks up significant peaks in the tails.

chosen S_3 :	$\{x_1, x_2, x_3\}$	$\{x_1, x_2, x_4\}$	$\{x_1, x_3, x_4\}$	$\{x_2, x_3, x_4\}$
$D(p \hat{p})$:	3.5487	1.1029	0.5373	2.9430
$\hat{D}(p \hat{p})$:	3.5487	1.1044	0.4263	2.8230
$ p - \hat{p} $:	0.1000	0.1000	0.3500	1.2482

Table 2.1: Distances between $p(x)$ and $\hat{p}(x)$ for different particle set S_3 .

This idea is better demonstrated in figure 2.16. Figure 2.16.a shows $\log p(\mathbf{x}) = -E(\mathbf{x})$ which is renormalized for displaying. $p(\mathbf{x})$ consists of $N = 50$ particles whose centers are shown by the black spots. The energies $E(\mathbf{x}_i), i = 1, 2, \dots, N$ are uniformly distributed in an interval $[0, 100]$. Thus their weights differ in exponential order. Figure 2.16.b shows $\log \hat{p}(\mathbf{x})$ with $k = 6$ particles that minimize both $D(p||\hat{p})$ and $\hat{D}(p||\hat{p})$. Figure 2.16.c shows the 6 particles that minimize the absolute difference $|p - \hat{p}|$. Figure 2.16.b has more disperse particles.

For segmentation, a remaining question is: how do we measure the distance between two solutions \mathbf{x}_1 and \mathbf{x}_2 ? This distance measure is to some extent subjective. We adopt a distance measure which simply accumulates the differences for the number of regions in $\mathbf{x}_1, \mathbf{x}_2$, and the types ℓ of image models used at each pixel by \mathbf{x}_1 and \mathbf{x}_2 .

2.6 Monte Carlo Tree Search

Chapter 3

Markov Chain Monte Carlo - the Basics

A **Markov Chain** is a mathematical model for stochastic systems whose states, *discrete* or *continuous*, are governed by a transition probability P . The current state in a Markov Chain only depends on the most recent previous states, *e.g.* for a 1st order Markov Chain

$$X_t | X_{t-1}, \dots, X_0 \sim P(X_t | X_{t-1}, \dots, X_0) = P(X_t | X_{t-1})$$

The **Markovian property** means “locality” in space or time, such as for Markov random fields and Markov Chain. Indeed, a discrete time Markov chain can be viewed as a special case of a Markov random field (causal and 1-dimensional).

A Markov Chain is often denoted by

$$\text{MC} = (\Omega, \nu, K)$$

where Ω is the state space, $\nu : \Omega \rightarrow \mathbb{R}$ is the initial probability distribution over the states, $K : \Omega \times \Omega \rightarrow \mathbb{R}$ is the transition probability.

Suppose that Ω is countable (better finite), then K is the matrix of transition probabilities $K(X_{t+1} | X_t)$. At time n , the MC state will follow a probability,

$$\nu_n = \nu_0 K^n$$

Example 3.1. Suppose we have a finite state space, $|\Omega| = N \sim 10^{30}$, then the transition probability K would be represented by a $N \times N$ transition matrix

$$K(X_{t+1} | X_t) = \begin{bmatrix} k_{11} & \cdots & k_{N1} \\ \vdots & \ddots & \vdots \\ k_{1N} & \cdots & k_{NN} \end{bmatrix}_{(N \times N)}$$

The transition matrix is usually sparse, but not always.

Thus, in SMC we try to design the trial probability $g(x)$, and in MCMC we will try to design the transition matrix $K(X_{t+1} | X_t)$. Therefore,

$$X_n \sim \underbrace{(\cdots)_{(1 \times N)}}_{\nu_n} = \underbrace{(\cdots)_{(1 \times N)}}_{\nu_{n-1}} \begin{bmatrix} k_{11} & \cdots & k_{N1} \\ \vdots & \ddots & \vdots \\ k_{1N} & \cdots & k_{NN} \end{bmatrix}_{(N \times N)}$$

MCMC is a **general purpose technique** for generating **fair samples** from a probability in a *high-dimensional* space, driven by random numbers drawn from a uniform probability in $[a, b]$. A Markov Chain is designed to have a p.d.f. $\pi(x)$ as its **stationary (invariant) probability**.

Many stochastic systems in physics, chemistry, economics can be simulated by MCMC.

Example 3.2. Suppose there are 5 families in an island. There are 1,000,000 tokens as their currency, and we normalize them to 1. Let \mathbf{x} be a 5×1 vector for the wealth of the 5 families over the years. Each family will trade with other families for goods. For example, family 1 will spend 60% of their income to buy from family 2, and save 40% income, and so on, as shown in Figure 3.1. The question is: how will the fortune be distributed among the families after a number of years? To ask in a different way, suppose we mark one token in a special color (say, red), after m years, who will own this token?

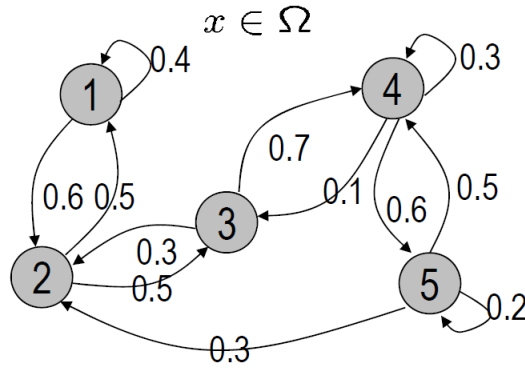


Figure 3.1: Trade diagram for five families.

We convert this to a mathematical model. Denote the state space for the red token to be $\Omega = \{1, 2, 3, 4, 5\}$.

Then the transition Kernel is

$$K = \begin{pmatrix} 0.4 & 0.6 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.3 & 0.0 & 0.7 & 0.0 \\ 0.0 & 0.0 & 0.1 & 0.3 & 0.6 \\ 0.0 & 0.3 & 0.0 & 0.5 & 0.2 \end{pmatrix}$$

Computing the distribution of wealth starting from different initial conditions we get:

Under certain conditions for the finite state Markov chains, the Markov chain state converges to an invariant probability.

$$\lim_{n \rightarrow \infty} \nu_0 K^n = \pi$$

In Bayesian inference, we are given a target probability π , our objective is to design a Markov chain kernel K , so that K has a *unique* invariant probability π .

In general, there are infinitely many K 's that have the same invariant probability.

$$\begin{array}{ccccccc} X_1 & \longrightarrow & X_2 & \longrightarrow & \cdots & \longrightarrow & X_n & \longrightarrow \\ \wr & & \wr & & & & \wr & \wr \\ \nu_1 & & \nu_2 & & \cdots & & \nu_n & \pi \end{array}$$

Suppose we are given Ω and a target probability $\pi = (\pi_1, \dots, \pi_N)_{(1 \times N)}$, our goal is to design ν_0 and K so that

Table 3.1: Final wealth distribution after convergence in Example 3.2 when starting from different initial distributions.

Year	A					B				
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.4	0.6	0.0	0.0	0.0	0.0	0.3	0.0	0.7	0.0
3	0.46	0.24	0.30	0.0	0.0	0.15	0.0	0.22	0.21	0.42
4				
5				
6	0.23	0.21	0.16	0.21	0.17	0.17	0.16	0.16	0.26	0.25
...				
Final	0.17	0.20	0.13	0.28	0.21	0.17	0.20	0.13	0.28	0.21

1) $\pi K = \pi$;

This is a necessary condition for the Markov chain to have the stationary probability π .

2) Fast convergence. Fast convergence can be obtained using a:

- good initial probability ν_0
- good transition matrix K

Usually the transition matrix is sparse (zero almost everywhere) due to the local connectivity, since the new state of a MCMC move is usually near the current state. There are some isolated examples where this is not the case, such as the Swendsen-Wang algorithm from Chapter 6.

3.1 Topology of Transition Matrix: Communication and Period

Now, we check the conditions for MC design.

1. Stochastic Matrix. The kernel matrix K should be a stochastic matrix, i.e.

$$\sum_{j=1}^N K_{ij} = 1, \quad \forall i \in \Omega, \quad K_{ij} \geq 0$$

or in matrix form:

$$K\mathbf{1} = \mathbf{1}$$

where $\mathbf{1}$ is the $N \times 1$ matrix of 1's: $\mathbf{1} = (1, \dots, 1)^T$.

2. Global Balance. Another necessary condition is global balance:

$$\pi K = \pi \quad \rightarrow \quad \sum_{i=1}^N \pi_i K_{ij} = \pi_j \quad \forall j \in \Omega$$

These conditions can be replaced by the *detailed balance* condition (a sufficient but not necessary condition):

$$\pi(i)K_{ij} = \pi(j)K_{ji}, \quad \forall i, j \in \Omega \quad (3.1)$$

Indeed, detailed balance implies stationarity:

$$\begin{aligned}\pi K &= \sum_i \pi(i) K_i = \sum_i \pi(i) (K_{i1}, \dots, K_{iN}) \\ &= \sum_i \left(\pi(1) K_{1i}, \dots, \pi(N) K_{Ni} \right) = \pi\end{aligned}$$

and in particular global balance

$$\sum_i \pi(i) K_{ij} = \sum_i \pi(j) K_{ji} = \pi(j) \sum_i K_{ji} = \pi(j)$$

A kernel that satisfied the detailed balance condition is called *reversible*.

Going back to Example 3.2 we can get the intuition that the global balance equation represents the total wealth conservation. Indeed, the total amount received by family j is $\sum_i \pi(i) K_{ij}$ and it is equal to family j 's wealth $\pi(j)$, which is the amount spent by family j .

There are infinite ways to construct K given a π . In global balance, we have $2N$ equations with $N \times N$ unknowns, in detailed balance we have $\frac{N^2}{2} + N$ equations with $N \times N$ unknowns.

3. Irreducibility. A state j is said to be accessible from state i if there exists M such $(K^M)_{ij} > 0$

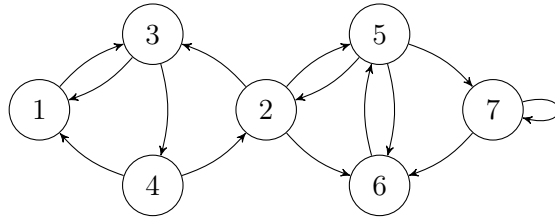
$$i \rightarrow j \quad (K^M)_{ij} = \sum_{i_1, \dots, i_{M-1}} K_{ii_1} \cdots K_{i_{M-1}j} > 0$$

If i and j are accessible to each other then we write $i \leftrightarrow j$. The communication relation \leftrightarrow generates a partition of the state space into disjoint equivalence classes called **communication classes**.

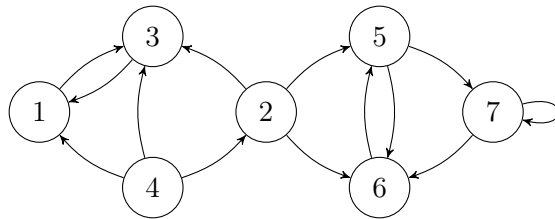
$$\Omega = \cup_{i=1}^C \Omega_i$$

Definition 3.1. A Markov Chain is *irreducible* if its transition matrix K has only $C = 1$ communication class.

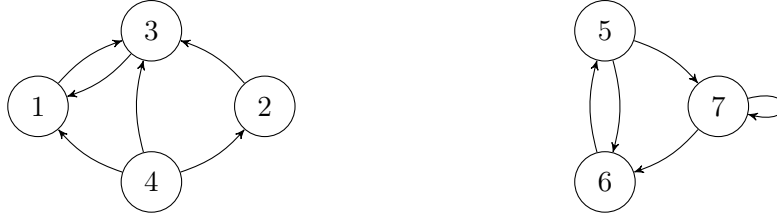
Example 3.3. Irreducible MC:



Example 3.4. Reducible MC:



⇓



In general a greedy optimization algorithm gets stuck in a local optimum and is an example of a reducible chain.

Given a distribution π , the ideal transition kernel would be $K = \begin{pmatrix} \pi \\ \pi \\ \dots \\ \pi \end{pmatrix}$, which always converges

in one step no matter where it starts. However, in general it is difficult to sample the distribution π directly so this kernel is not very useful in practice.

4. Aperiodicity. To define aperiodicity, we first need to define what is a periodic Markov chain.

Definition 3.2. An irreducible Markov chain with transition matrix K has period d if one can find a (unique) partition of graph G into d cyclic classes:

$$C_1, \dots, C_d, \quad \sum_{j \in C_k} K_{ij} = 1, \quad \forall i \in C_{k-1},$$

Remark 3.1. In a periodic Markov chain there is no connection between states within each individual cyclic class. The transition matrix is a block matrix of the following form.

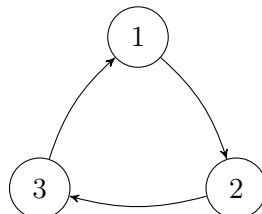
$$K = \begin{pmatrix} & & \blacksquare & & \\ & & & \blacksquare & \\ & & & & \blacksquare \\ \blacksquare & & & & \\ & & & & \end{pmatrix}$$

Then, the transition matrix of K^d becomes a diagonal block matrix

$$K^d = \begin{pmatrix} \blacksquare & & & & \\ & \blacksquare & & & \\ & & \blacksquare & & \\ & & & \blacksquare & \\ & & & & \blacksquare \end{pmatrix}$$

This means that K has one communication class, but K^d has d communication classes.

Example 3.5. Consider the Markov chain with the following transition kernel: $K = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$



It has period 3 and alternates three distributions: $(1\ 0\ 0) \rightarrow (0\ 1\ 0) \rightarrow (0\ 0\ 1)$.

Definition 3.3. An irreducible Markov chain with transition matrix K is aperiodic if its largest period is $d = 1$.

5. Stationary distribution. A Markov chain with transition kernel K has stationary distribution π if

$$\pi K = \pi$$

There may be many stationary distributions w.r.t K . Even there is a stationary distribution, a Markov chain may not always converge to it.

Example 3.6. Consider the Markov chain with the transition kernel: $K = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$. It has $\pi = (\frac{1}{3} \ \frac{1}{3} \ \frac{1}{3})$ as stationary distribution but it might never converge to it, as seen in Example 3.5.

3.2 The Perron-Frobenius Theorem

Theorem 3.4 (Perron-Frobenius). *For any primitive (irreducible and aperiodic) $N \times N$ stochastic matrix K , K has eigenvalues*

$$1 = \lambda_1 > |\lambda_2| > \dots > |\lambda_r|$$

with multiplicities m_1, \dots, m_r and left and right eigenvectors $(\mathbf{u}_i, \mathbf{v}_i)$ respectively. Then $\mathbf{u}_1 = \pi, \mathbf{v}_1 = \mathbf{1}$, and

$$K^n = \mathbf{1} \cdot \pi + O(n^{m_2-1} |\lambda_2|^n).$$

We define $\lambda_{\text{slem}} = |\lambda_2|$, which is the second largest eigenvalue modulus. Then obviously, the convergence rate is decided by λ_{slem} .

Remark 3.2. If K is not irreducible, and has C communication classes, then K is block diagonal with C blocks. Consequently the eigenvalue 1 has at least C distinct eigenvectors, and K does not have a unique invariant probability.

Remark 3.3. If K is irreducible but has period $d > 1$, then there it has at least d distinct eigenvalues with modulus 1, namely, the d^{th} roots of unity. This is because its characteristic polynomial is $\det(tI - K) = \det(t^d I - K_1 \dots K_d)$ as it can be proved by induction, where K_1, \dots, K_d are the non-zero blocks. But K_1, \dots, K_d all have eigenvalue 1, so $U = K_1 \dots K_d$ has eigenvalue 1, so its characteristic polynomial $\det(tI - U)$ is divisible by $t - 1$, therefore $\det(tI - K) = \det(t^d I - U)$ is divisible by $t^d - 1$.

Review

Suppose K is a $N \times N$ positive non-symmetric matrix, and has N eigenvalues.

$$\begin{array}{ccccccc} \lambda_1 & & \dots & & \lambda_N \\ & \swarrow & & \searrow & & \swarrow & \searrow \\ & u_1 & & v_1 & & u_N & v_N \end{array}$$

Each eigenvalue has a corresponding right and left eigenvector. λ, u, v are all complex numbers.

$$\begin{aligned} u_i K &= \lambda_i u_i, & u_i &: 1 \times N \\ K v_i &= \lambda_i v_i, & v_i &: N \times 1 \end{aligned}$$

Therefore,

$$\begin{aligned} K &= \lambda_1 v_1 u_1 + \lambda_2 v_2 u_2 + \dots + \lambda_N v_N u_N \\ K \cdot K &= \sum_{i=1}^N \lambda_i v_i u_i \cdot \sum_{j=1}^N \lambda_j v_j u_j = \sum_{\substack{i=1 \\ j=1}}^N \lambda_i \lambda_j v_i u_i v_j u_j, & \begin{cases} \text{if } i \neq j & u_i v_j = 0 \\ \text{if } i = j & u_i v_j = 1 \end{cases} \end{aligned}$$

Thus,

$$K^n = \lambda_1^n v_1 u_1 + \lambda_2^n v_2 u_2 + \dots + \lambda_N^n v_N u_N$$

Since we have global balance,

$$\begin{aligned} \pi K = \pi & \implies \lambda_1 = 1, u_1 = \pi \\ K \mathbf{1} = \mathbf{1} & \implies \lambda_1 = 1, v_1 = \mathbf{1} \end{aligned} \bigg\} \implies \lambda_1 \cdot v_1 \cdot u_1 = \begin{pmatrix} \pi \\ \pi \\ \dots \\ \pi \end{pmatrix}$$

Thus,

$$K^n = \begin{pmatrix} \pi \\ \pi \\ \dots \\ \pi \end{pmatrix} + \underbrace{\epsilon}_{\rightarrow 0} \quad \text{if } |\lambda_i| < 1, \forall i > 1$$

so K^n approaches the ideal kernel $\begin{pmatrix} \pi \\ \pi \\ \dots \\ \pi \end{pmatrix}$ as $n \rightarrow \infty$.

3.3 Convergence Measures

One thing we are interested in is the state i which is the global optimum of the probability,

$$i^* = \operatorname{argmax} \pi(x)$$

Definition 3.5. Given a Markov chain (x_0, \dots, x_n, \dots) with transition kernel K and invariant probability π we define

i) The first hitting time of a state i (in the finite state case)

$$\tau_{\text{hit}}(i) = \inf\{n \geq 1; x_n = i, x_0 \sim \nu_0\}, \quad \forall i \in \Omega$$

$E[\tau_{\text{hit}}(i)]$ is the mean first hitting time of i for the Markov chain governed by K .

ii) First return time of a state i

$$\tau_{\text{ret}}(i) = \inf\{n \geq 1; x_n = i, x_0 = i\}, \quad \forall i \in \Omega$$

iii) Mixing time

$$\tau_{\text{mix}} = \min_n \{\|\nu_0 K^n - \pi\|_{\text{TV}} \leq \epsilon, \forall \nu_0\}$$

where the total variation is defined as

$$\|\mu - \nu\|_{\text{TV}} = \frac{1}{2} \sum_{i \in \Omega} |\mu(i) - \nu(i)| = \sum_A \left(\mu(i) - \nu(i) \right), \quad A = \{i : \mu(i) \geq \nu(i), i \in \Omega\}$$

Definition 3.6. The contraction coefficient for K is the maximum TV-norm between any two rows in the transition kernel and is calculated by

$$C(K) = \max_{x,y} \|K(x, \cdot) - K(y, \cdot)\|_{\text{TV}}$$

Example 3.7. Consider the Markov kernel for the five families living in an island, where the values are different than in example 3.2

$$K = \begin{pmatrix} 0.3, & 0.6, & 0.1, & 0.0, & 0.0 \\ 0.2 & 0.0, & 0.7, & 0.0, & 0.1 \\ 0.0, & 0.5, & 0.0, & 0.5, & 0.0 \\ 0.0, & 0.0, & 0.4, & 0.1, & 0.5 \\ 0.4, & 0.1, & 0.0, & 0.4, & 0.1 \end{pmatrix}$$

1) We plot in Figure 3.2, left, the five complex eigenvalues, in a 2D plane. The invariant

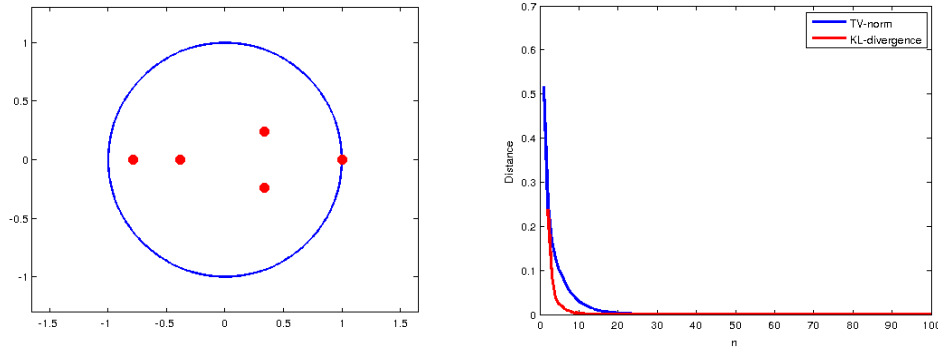


Figure 3.2: The five families kernel of Example 3.7. Left: the five complex eigenvalues. Right: the TV norm and KL divergence between $\mu_n = \nu \cdot K^n$ and the invariant probability π .

probability is $\pi = (0.1488 \ 0.2353 \ 0.2635 \ 0.2098 \ 0.1427)$. The second largest eigenvalue has $\lambda_{slem} = \|\lambda_2\| = 0.7833$.

2) Suppose we start with an initial probabilities $\nu = (1, 0, 0, 0, 0)$ i.e. we know for sure that the initial state is at $x_0 = 1$. So, at step n , the Markov chain state follows a distribution $\mu_n = \nu \cdot K^n$. We compute the distance between μ_n and π using the TV-norm,

$$d_{\text{TV}}(n) = \|\pi - \mu_n\|_{\text{TV}} = \frac{1}{2} \sum_{i=1}^5 |\pi(i) - \mu_n(i)|;$$

or KL-divergence,

$$d_{\text{KL}}(n) = \sum_{i=1}^5 \pi(i) \log \frac{\pi(i)}{\mu_n(i)}.$$

The plot of the two distances $d_{\text{TV}}(n)$ and $d_{\text{KL}}(n)$ for the first 1000 steps is shown in Figure 3.2, right.

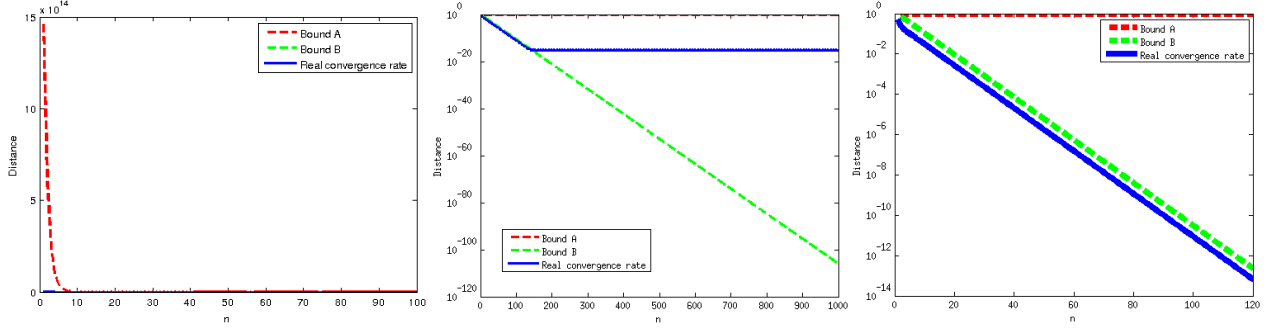


Figure 3.3: The TV norm between $\mu_n = \nu \cdot K^n$ and the invariant probability π and the two bounds $A(n)$ and $B(n)$ from Eq. (3.2) and (3.3) respectively. Left: original scale. Center: log-scale. Right: zoom-in of the first 120 iterations.

3) We calculate the contraction coefficient for K . Note that contraction coefficient is the maximum TV-norm between any two rows in the transition kernel,

$$C(K) = \max_{x,y} \|K(x, \cdot) - K(y, \cdot)\|_{\text{TV}}.$$

One can prove that

$$\|\nu_1 \cdot K - \nu_2 \cdot K\|_{\text{TV}} \leq C(K) \|\nu_1 - \nu_2\|_{\text{TV}}$$

As $\|\nu_1 - \nu_2\|_{\text{TV}} \leq 1$, if $C(K) < 1$ then the convergence rate could be upper bounded by

$$\|\nu_1 \cdot K^n - \nu_2 \cdot K^n\|_{\text{TV}} \leq C^n(K) \|\nu_1 - \nu_2\|_{\text{TV}} \leq C^n(K) = A(n), \quad \forall \nu_1, \nu_2. \quad (3.2)$$

But for this example, we can see that $C(K) = 1$ so the bound is not very good.

4) There is another bound – the Diaconis-Hanlon bound below,

$$\|\pi - \nu K^n\|_{\text{TV}} \leq \sqrt{\frac{1 - \pi(x_0)}{4\pi(x_0)}} \lambda_{\text{slcm}}^n = B(n) \quad (3.3)$$

where $x_0 = 1$ is the initial state and $\pi(x_0)$ is a target probability at $x = 1$. The real convergence rate $d_{\text{TV}}(n)$ is plotted in comparison with $A(n)$ and $B(n)$ in Figure 3.3, left on the original scale and center on the log-scale. The bound only holds until the machine precision is attained. In Figure 3.3, is shown a zoom-in of the log-scale for the first 120 iterations.

3.4 Markov Chains in Continuous or Heterogeneous State spaces

In continuous case, the target $\pi : \Omega \rightarrow \mathbb{R}$ is a p.d.f. $\pi(x)$ and the transition kernel is a conditional p.d.f. $K(x, y) = K(y|x)$, so $\int_{\Omega} K(x, y) dy = 1$.

The global balance equation must be satisfied for any event $A \subseteq \Omega$,

$$\pi K(A) = \int_A \int_{\Omega} \pi(x) K(x, y) dx dy = \int_A \pi(x) dx = \pi(A)$$

The detailed balance equation in the continuous case is

$$\int_A \int_B \pi(x) K(x, y) dx dy = \int_A \int_B \pi(y) K(y, x) dx dy$$

In practice, Ω is a mixed/heterogeneous space: discrete/finite and continuous variables.

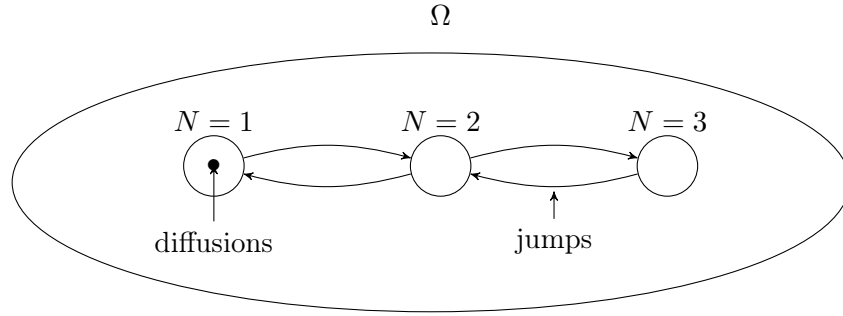


Figure 3.4: Illustration of jump-diffusion processes for the heterogeneous space from Example 3.8.

Example 3.8. The heterogeneous space of $X = \{N, (x_i, y_i), i = 1, \dots, N\}$, where N is the number of people in a picture and (x_i, y_i) are their positions.

In such a case there are many different MC processes, as illustrated in Figure 3.4. Irreducible MCMC will have many dynamics(sub chains) such as:

$$\left\{ \begin{array}{l} \text{Jumps} \left\{ \begin{array}{l} \text{Death/Birth} \\ \text{Split/Merge} \end{array} \right\} \text{process} \\ \text{Diffusions} \end{array} \right.$$

3.5 Ergodicity Theorem

Definition 3.7. A state i is said to be a *recurrent state* if it has

$$P(\tau_{\text{ret}}(i) < \infty) = 1,$$

otherwise it is a *transient state*. The quantity $\tau_{\text{ret}}(i)$ is the “return” time, the total steps to return from x to x .

Definition 3.8. A state i that satisfies

$$E[\tau_{\text{ret}}(i)] < \infty$$

is called a *positively recurrent state*, otherwise it is a null-recurrent state. A Markov chain is positively recurrent if all of its state are positively recurrent.

Usually, positive recurrence is a condition for spaces with infinite states.

Theorem (Ergodicity theorem). *For an irreducible, positively recurrent Markov Chain with stationary probability π , in a state space Ω , let $f(x)$ be any real valued function with finite mean with respect to π , then for any initial probability, almost surely we have*

$$\lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=1}^N f(x_i) = \sum_{x \in \Omega} f(x) \pi(x) = E_{\pi}[f(x)], \quad \forall \text{ function } f$$

where $x_i \sim MC$ states (but do not need to be i.i.d.).

3.6 MCMC for Optimization by Simulated Annealing

An MCMC algorithm is designed to obtain samples from a posterior distribution π , $X \sim \pi$. We saw that under certain conditions (detailed balance, irreducibility and aperiodicity) the Markov chain invariant probability will converge to the stationary distribution π after a burn-in period.

MCMC can also be used for optimization by slowly changing the stationary distribution π while running the Markov chain. Suppose we want to minimize a function $f(x) : \Omega \rightarrow \mathbb{R}$. Then we consider the posterior probability

$$\pi(x; T) = \frac{1}{Z(T)} \exp(-f(x)/T)$$

that depends on a parameter T which is called “temperature”. When T is large, the probability $\pi(x, T)$ will have smaller peaks and local maxima and it will be easier to sample from it. When T is very small, the probability $\pi(x, T)$ will be concentrated at its global maximum, as illustrated in Figure 3.5.

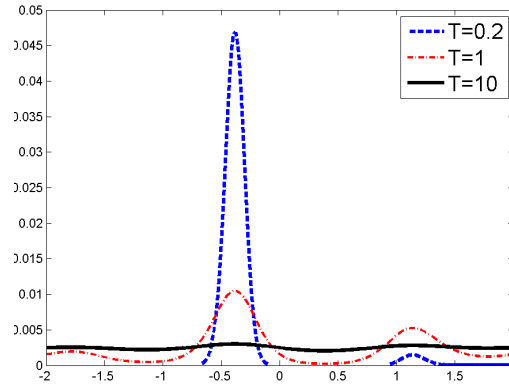


Figure 3.5: The influence of the temperature on a probability distribution. At temperature $T = 10$ the probability is close to uniform, while at $T = 0.1$ the global optimum is clearly visible.

The annealing procedure means running the Markov chain at a high temperature and slowly decreasing the temperature while running the MC until a very small temperature is reached. This procedure is inspired from the annealing method for producing crystalline structures in metals or other materials. There the material is heated to a high temperature where it melts, then the temperature is slowly decreased to allow the atoms to position themselves in a low energy configuration, which gives the crystalline structure. If the material is cooled too quickly, it will develop cracks or other defects and it will have small crystals. If the material cooling is slow enough, larger crystals with fewer defects can be obtained.

Similarly, for optimization one needs to select an *annealing schedule* T_k that specifies the temperature used at each step of the Markov chain. The schedule starts with a high T_0 and decreases to 0 as $k \rightarrow \infty$, so $\lim_{k \rightarrow \infty} T_k = 0$. Through this annealing schedule, the probability $\pi(x, T)$ becomes $\pi(x, T_k)$, a time varying probability. The simulated annealing algorithm is described in Algorithm 1 below, where for any $x \in \Omega$, $N(x)$ is the set of possible states that are accessible from state x in one Markov chain step.

```

input: Initial solution  $x \in \Omega$ 
input: Temperature cooling schedule,  $T_k$ 
input: Initial temperature  $T = T_0 > 0$ 
input: Repetition schedule  $M_k$ , the number of iterations executed at each temperature  $T_k$ 
set the temperature change counter  $k = 0$ 
repeat
  for  $m = 0$  to  $M_k$  do
    generate a solution  $x' \in N(x)$ 
    calculate  $\Delta_{x,x'} = f(x') - f(x)$ 
    if  $\Delta_{x,x'} \leq 0$  then
       $x \leftarrow x'$ 
    else
       $x \leftarrow x'$  with probability  $\exp(-\Delta_{x,x'}/T_k)$ 
    end if
  end for
   $k = k + 1$ 
until stopping criterion is met

```

Algorithm 1: Simulated Annealing

There are two types of convergence results, based on modeling Algorithm 1 as a sequence of homogeneous Markov chains or as a single inhomogeneous Markov chain.

The following is an inhomogeneous Markov chain result due to Mitra [141].

Theorem 3.4. (Mitra 1986) The Markov chain associated with simulated annealing with the following update function

$$T_k = \frac{\gamma}{\log(k + k_0 + 1)}$$

for any parameter $k_0 \geq 1$ and γ sufficiently large converges to a global optimum starting from any initial solution $x \in \Omega$.

In reality, one cannot wait so long to find a solution and faster annealing schedules are used in practice, decreasing linearly or even exponentially in t . With these schedules, the optimization algorithm finds a local optimum that can be good or bad depending on the annealing schedule and the MCMC algorithm used. Some MCMC algorithms such as the Gibbs sampler need slow annealing schedules to obtain good solutions while other algorithms such as the Swendsen-Wang Cut allow faster colling schedules to obtain similar solutions.

Remark 3.5. In many computer vision problems finding the global optimum of $\pi(x)$ can be NP hard, which means a polynomial optimization algorithm should not be expected to be found. In these cases, the annealing schedule will have to decrease logarithmically in t so the global optimum will be found in exponential time on the problem size (e.g. dimensionality of Ω).

Chapter 4

Metropolis Methods and Variants

4.1 The Metropolis-Hastings Algorithm

The Metropolis Algorithm [139,140] has been declared number 1 in the list of top ten algorithms of the 20th century by Dongarra and Sullivan [57]. The original algorithm [139] has been proposed for equations of state in chemical physics and has been generalized to its current form by Hastings [91].

The Metropolis-Hastings algorithm is a simple method to take any algorithm that tries to jump from a current state X to a new state Y and slightly modify it by accepting the move with a probability in order for the resulting algorithm to satisfy the detailed balance equation (3.1).

Example 4.1. The idea of Metropolis-Hasting algorithm is illustrated in Figure 4.1. Suppose that there is a "proposed" algorithm that tries to move between states X and Y according to the probabilities displayed in Figure 4.1, left.

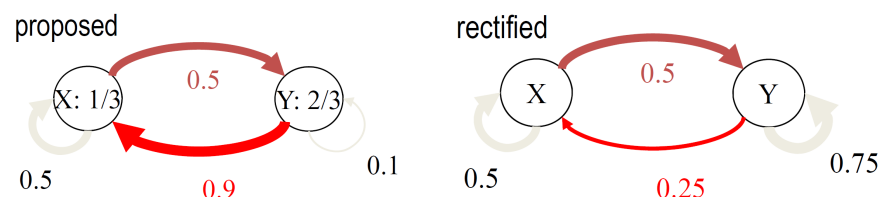


Figure 4.1: Illustration of the Metropolis-Hastings algorithm. Left: the proposed move between states X and Y does not satisfy detailed balance. Right: the transition probabilities are rectified to satisfy the detailed balance equation.

Since $\pi(X) = 1/3$ and $\pi(Y) = 2/3$, the detailed balance equation is

$$K(X, Y) \frac{1}{3} = K(Y, X) \frac{2}{3}$$

and it is easy to check that it is not satisfied under the proposed transition probabilities.

The move between X and Y is rectified with an acceptance probability, $\alpha = \frac{0.5 \times \frac{1}{3}}{0.9 \times \frac{2}{3}} = \frac{5}{18}$. Only $\frac{5}{18}$ of the proposal from Y to X is allowed, and all proposals from X to Y are allowed. The acceptance probability rectifies the proposal probability, so that the MC follows the target distribution. The rectified probabilities are shown in Figure 4.1, right.

4.1.1 The original Metropolis-Hastings algorithm

The Metropolis Hastings algorithm resembles importance sampling in the sense that it uses a simpler distribution $Q(x, y)$ to generate proposal samples, which are then reweighed by an acceptance probability. In general the proposal distribution $Q(x, y)$ is simple and samples of y conditional on x can be obtained easily.

Input: Target probability distribution $\pi(x)$, current state $x^{(t)} \in \Omega$, and proposal probability distribution $Q(x, y)$.

Output: New state $x^{(t+1)} \in \Omega$

1. Propose a new state y by sampling from $Q(x^{(t)}, y)$.
2. Compute the acceptance probability:

$$\alpha(x, y) = \min \left(1, \frac{Q(y, x)}{Q(x, y)} \cdot \frac{\pi(y)}{\pi(x)} \right) \quad (4.1)$$

3. With probability $\alpha(x, y)$ accept the move and make $x^{(t+1)} = y$, otherwise $x^{(t+1)} = x^{(t)}$.

Figure 4.2: **One step of the Metropolis-Hastings Algorithm**

Theorem 4.1 (Metropolis-Hastings). *The Metropolis-Hastings algorithm from Figure 4.2 satisfies the detailed balance equation.*

Proof. We have

$$\underbrace{K(x, y)}_{\text{transition probability}} = \begin{cases} \underbrace{Q(x, y)}_{\text{proposal}} \cdot \underbrace{\alpha(x, y)}_{\text{acceptance rate}} = Q(x, y) \cdot \min \left(1, \underbrace{\frac{Q(y, x)}{Q(x, y)}}_{\text{proposal}} \cdot \underbrace{\frac{\pi(y)}{\pi(x)}}_{\text{verification}} \right), & \forall y \neq x \\ 1 - \sum_{y \neq x} Q(x, y) \alpha(x, y), & y = x \end{cases}$$

Since,

$$\alpha(x, y) = \min \left(1, \frac{Q(y, x)}{Q(x, y)} \cdot \frac{\pi(y)}{\pi(x)} \right)$$

$$\alpha(y, x) = \min \left(1, \frac{Q(x, y)}{Q(y, x)} \cdot \frac{\pi(x)}{\pi(y)} \right)$$

we will have either $\alpha(x, y) = 1$ or $\alpha(y, x) = 1$. Thus, for detailed balance, the left hand side will be

$$\pi(x)K(x, y) = \pi(x)Q(x, y)\alpha(x, y) = \pi(x)Q(x, y) \min \left(1, \frac{Q(y, x)}{Q(x, y)} \cdot \frac{\pi(y)}{\pi(x)} \right) = \min \left(\pi(x)Q(x, y), \pi(y)Q(y, x) \right)$$

and the right hand side will be

$$\pi(y)K(y, x) = \pi(y)Q(y, x)\alpha(y, x) = \pi(y)Q(y, x) \min \left(1, \frac{Q(x, y)}{Q(y, x)} \cdot \frac{\pi(x)}{\pi(y)} \right) = \min \left(\pi(x)Q(x, y), \pi(y)Q(y, x) \right)$$

therefore, the detailed balance equation is satisfied. \square

4.1.2 Another version of the Metropolis-Hastings algorithm

In many cases the target probability is written as a Gibbs distribution:

$$\pi(x) = \frac{1}{Z} e^{-E(x)}$$

and the normalization constant is hard to compute. Suppose the proposal probability is symmetric: $Q(x, y) = Q(y, x)$. Then the acceptance probability becomes:

$$\alpha(x, y) = \min \left(1, \frac{\pi(x)}{\pi(y)} \right) = \min (1, e^{-(E(x)-E(y))}) = \min(1, e^{-\Delta E})$$

Thus,

$$\begin{aligned} \alpha(x, y) &= 1, & \text{if } \Delta E < 0, & \text{ i.e. } y \text{ is a lower energy state (better) than } x \\ \alpha(x, y) &= e^{-\Delta E} < 1, & \text{if } \Delta E > 0, & \text{ i.e. } y \text{ is a higher energy state (worse) than } x \end{aligned}$$

ΔE is often computed locally as the two states x and y share most of the elements. When the proposal is rejected (with probability $1 - \alpha$), the MC stays at state x .

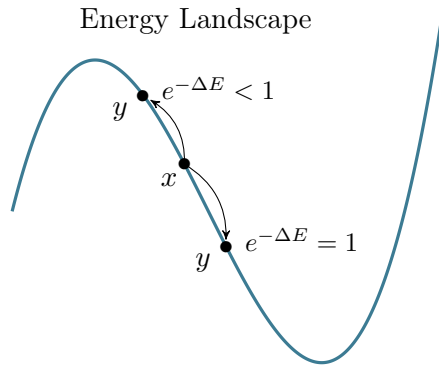


Figure 4.3: Illustration of the Metropolis-Hastings algorithm variant for Gibbs distributions.

The procedure is illustrated in Figure 4.3. Note that $Q(y, x)$ is designed to make informed proposals that could guide the Markov chain in the right direction.

Remark 4.1. We must be careful with the assumption $Q(x, y) = Q(y, x)$, since it is usually violated at the boundary of the domain Ω .

4.1.3 Other acceptance probability designs

There exist other designs for the acceptance rate that guarantee the detailed balance equation, such as

$$\alpha(x, y) = \frac{\pi(y)Q(y, x)}{\pi(y)Q(y, x) + \pi(x)Q(x, y)}$$

or more generally

$$\alpha(x, y) = \frac{s(x, y)}{\pi(x)Q(x, y)}$$

where $s(x, y)$ is any symmetric function.

Remark 4.2. Going back to the Example 3.2, one could think of the $\pi(x)$ as the equilibrium distribution of wealth among a number of families that frequently trade with each other. The proposals $Q(x, y)$ can be regarded as trade proposals between the families. In this context, the Metropolis-Hastings choice of acceptance probability maximizes the trade between families among all designs based on $Q(x, y)$ that satisfy detailed balance.

4.1.4 Key issues in Metropolis design

Intuitively, the MH method allows a probability to climb out of the local minima. The key issue in designing the Metropolis algorithm is the design of the proposal probability $Q(x, y)$. Some desirable properties of $Q(x, y)$ are:

- i. For any x , the set of reachable states, $\{y, Q(x, y) > 0\}$ is large, so $K(x, y)$ is more connected.
- ii. For any x , the probability $Q(x, y)$ is far from uniform (*i.e.* well-informed).

4.2 The Independence Metropolis Sampler

The IMS is a Metropolis-Hastings type algorithm with the proposal independent of the current state of the chain. It has also been called Metropolized Independent Sampling (Liu [122]). The goal is to simulate a Markov chain $\{X_m\}_{m \geq 0}$ taking values in Ω and having stationary distribution $\pi = (\pi_1, \pi_2, \dots, \pi_N)$ (the target probability), with a very large N , e.g. $N = 10^{20}$, when it's practically impossible to enumerate all the states. In this case, at each step a new state $j \in \Omega$ is sampled from the proposal probability $q = (q_1, q_2, \dots, q_N)$ according to $j \sim q_j$, which is then accepted with probability

$$\alpha(i, j) = \min\left\{1, \frac{q_i \pi_j}{\pi_i q_j}\right\}.$$

Therefore, the transition from X_m to X_{m+1} is decided by the transition kernel having the form

$$\mathcal{K}(i, j) = \begin{cases} q_j \alpha(i, j), & j \neq i, \\ 1 - \sum_{k \neq i} \mathcal{K}(i, k), & j = i. \end{cases}$$

The initial state could be either fixed or generated from a distribution whose natural choice in this case is q . In section 4.2.3, we show why it is more efficient to generate the initial state from q instead of choosing it deterministically.

It is easy to show that π is the invariant (stationary) distribution of the chain. In other words, $\pi \mathcal{K} = \pi$. Since from $q > 0$ it follows that \mathcal{K} is ergodic, then π is also the equilibrium distribution of the chain. Therefore, the marginal distribution of the chain at step m , for m large enough, is approximately π .

However, instead of trying to sample from the target distribution π , one may be interested in searching for a state i^* with maximum probability: $i^* = \arg \max_{i \in \Omega} \pi_i$. Here is where the mean first hitting time can come into play. $E[\tau(i)]$ is a good measure for the speed of search in general. As a special case we may want to know $E[\tau(i^*)]$ for the optimal state.

As it shall become clear later, a key quantity to the analysis is the probability ratio $w_i = q_i / \pi_i$. It measures how much knowledge the heuristic q_i has about π_i , or in other words how *informed* is q about π for state i . Therefore we define the following concepts.

Definition 4.2. A state i is said to be *over-informed* if $q_i > \pi_i$ and *under-informed* if $q_i < \pi_i$.

There are three special states defined below.

Definition 4.3. A state i is *exactly-informed* if $q_i = \pi_i$. A state i is *most-informed* (or *least-informed*) if it has the highest (or lowest) ratio $w_i = q_i/\pi_i$: $i_{\max} = \arg \max_{i \in \Omega} \{ w_i \}$, $i_{\min} = \arg \min_{i \in \Omega} \{ w_i \}$.

Liu [122] noticed that the transition kernel can be written in a simpler form by reordering the states increasingly according to their informedness. Since for $i \neq j$, $\mathcal{K}_{ij} = q_j \min\{1, w_i/w_j\}$, if $w_1 \leq w_2 \leq \dots \leq w_n$ it follows that

$$\mathcal{K}_{ij} = \begin{cases} w_i \pi_j & i < j, \\ 1 - \sum_{k < i} q_k - w_i \sum_{k > i} \pi_k & i = j, \\ q_j = w_j \pi_j & i > j. \end{cases}$$

Without loss of generality, we shall assume for the rest of the paper that the states are indexed such that $w_1 \leq w_2 \leq \dots \leq w_n$, to allow for this more tractable form of the transition kernel.

4.2.1 The eigenstructure of the IMS

In the last two decades a considerable amount of work has been devoted to studying properties of the IMS. Without trying to be comprehensive, we shall briefly review some of the results that were of interest to us. For finite state spaces, Diaconis and Hanlon [54] and Liu [122] proved various upper bounds for the total variation distance between updated and target distributions for the IMS. They showed that the convergence rate of the Markov chain is upper bounded by a quantity that depends on the second largest eigenvalue:

$$\lambda_{slem} = 1 - \min_i \left\{ \frac{q_i}{\pi_i} \right\}.$$

Remark 4.3. In the continuous case, denoting by $\lambda^* = 1 - \inf_x \{ \frac{q(x)}{p(x)} \}$, Mengersen and Tweedie [138] showed that if λ^* is strictly less than 1, the chain is uniformly ergodic, while if λ^* is equal to 1, the convergence is not even geometric anymore. Similar results were obtained by Smith and Tierney [176]. These results show that the convergence rate of the Markov chain for the IMS is subject to a *worst-case* scenario. For the finite case, the state corresponding to the least probability ratio q_i/π_i determines the rate of convergence. That is just one state from a potentially huge state space decides the rate of convergence of the Markov chain, and this state might be irrelevant to all the tasks of MCMC! A similar situation occurs in continuous spaces.

To illustrate this phenomenon, consider the following simple example.

Example 4.2. Let q and π be two Gaussians having equal variances and the means slightly shifted. Then q , as proposal distribution, will approximate the target π very well. However, it is easy to see that $\inf_x \{q(x)/p(x)\} = 0$ and therefore the IMS will not have a geometric rate of convergence.

This dismal behavior motivated our interest for studying the mean first hitting time as a measure of "speed" for Markov chains. This is particularly appropriate when dealing with stochastic search algorithms, when the focus could be on finding individual states rather than on the global convergence of the chain. For instance, in computer vision problems, one is often searching for the most probable interpretation of a scene and, to this end, various Metropolis-Hastings type algorithms can be employed. See Tu and Zhu [193] for examples and discussions. In such a context, it is of interest to find the behavior of the first hitting time of some states, like the modes of the posterior distribution of a scene given the input images.

4.2.2 General first hitting time for finite spaces

Consider an ergodic Markov chain $\{X_m\}_m$ on the finite space $\Omega = \{1, 2, \dots, n\}$. Let \mathcal{K} be the transition kernel, π its unique stationary probability, and q the starting distribution. For each state $i \in \Omega$, the *first hitting time* $\tau_{\text{hit}}(i)$ has been defined in section 3.3.

For any i , let us denote by \mathcal{K}_{-i} the $(n-1) \times (n-1)$ matrix obtained from \mathcal{K} by deleting the i^{th} column and row, that is, $\mathcal{K}_{-i}(k, j) = \mathcal{K}(k, j), \forall k \neq i, j \neq i$. Also let $q_{-i} = (q_1, \dots, q_{i-1}, q_{i+1}, \dots, q_n)$. Then, it is immediate that $P(\tau(i) > m) = q_{-i} \mathcal{K}_{-i}^{m-1} \mathbf{1}$, where $\mathbf{1} := (1, 1, \dots, 1)'$. This leads to the following formula for the expectation:

$$E_q[\tau(i)] = 1 + q_{-i}(\mathbf{I} - \mathcal{K}_{-i})^{-1} \mathbf{1}, \quad (4.2)$$

where \mathbf{I} denotes the identity matrix. The existence of the inverse of $\mathbf{I} - \mathcal{K}_{-i}$ is implied by the sub-stochasticity of \mathcal{K}_{-i} and the irreducibility of \mathcal{K} (Bremaud [25]).

More generally, the mean f.h.t of a subset A of Ω is given by

$$E_q[\tau(A)] = 1 + q_{-A}(\mathbf{I} - \mathcal{K}_{-A})^{-1} \mathbf{1}, \quad \forall A \subset \Omega. \quad (4.3)$$

4.2.3 Hitting time analysis for the IMS

Here, we shall capitalize on the previous result to compute the mean f.h.t for the IMS and provide bounds for it, by making use of the eigen-structure of the IMS kernel.

Theorem 4.4. (*Maciucă and Zhu, 2006*) Assume a Markov chain starting from q is simulated according to the IMS transition kernel having proposal q and target probability π . Then, using previous notations:

$$\begin{aligned} i) \quad & E[\tau(i)] = \frac{1}{\pi_i(1 - \lambda_i)}, \quad \forall i \in \Omega, \\ ii) \quad & \frac{1}{\min\{q_i, \pi_i\}} \leq E[\tau(i)] \leq \frac{1}{\min\{q_i, \pi_i\}} \frac{1}{1 - \|\pi - q\|_{TV}}, \end{aligned}$$

where we define λ_n to be equal to zero and $\|\pi - q\|_{TV}$ denotes the total variation distance between π and q . Equality is attained for the three special states from Definition 4.3.

The proof is in [127]. Theorem 4.4 can be extended by considering the first hitting time of some particular sets. The following corollary holds true, proved in [127].

Corollary 4.5. Let $A \subset \Omega$ of the form $A = \{i+1, i+2, \dots, i+k\}$, with $w_1 \leq w_2 \leq \dots \leq w_n$. Denote $\pi_A := \pi_{i+1} + \pi_{i+2} + \dots + \pi_{i+k}$, $q_A := q_{i+1} + q_{i+2} + \dots + q_{i+k}$, $w_A := q_A/\pi_A$ and $\lambda_A := (q_{i+1} + \dots + q_n) - (\pi_{i+1} + \dots + \pi_n)w_A$. Then *i)* and *ii)* from Theorem 4.4 hold ad-literam with i replaced by A .

In the introduction part we hinted at showing why generating the initial state from q is preferable to starting from a fixed state $j \neq i$. The following result attempts to clarify this issue.

Proposition 4.6. Assuming that $w_1 \leq w_2 \leq \dots \leq w_n$, the following inequalities hold true:

$$E_1[\tau(i)] \geq E_2[\tau(i)] \geq \dots \geq E_{i-1}[\tau(i)] \geq E_{i+1}[\tau(i)] = \dots = E_n[\tau(i)] = E[\tau(i)], \quad \forall i \in \Omega.$$

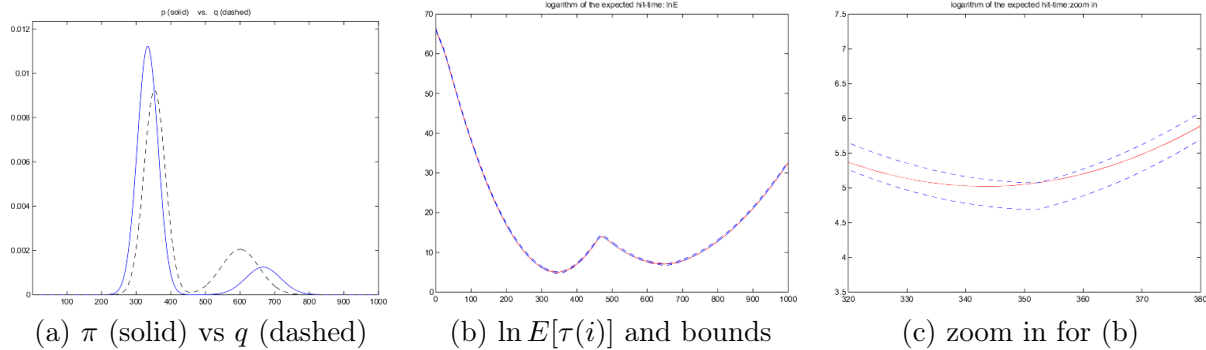


Figure 4.4: Mean first hitting time and bounds for Example 4.3

Example 4.3. We can illustrate the main results in Theorem 4.4 through a simple example. We consider a space with $n = 1000$ states. Let π and q be mixtures of two discretized Gaussians with tails truncated and then normalized to one. They are plotted as solid (π), dashed (q) curves in Fig.4.4a. Fig.4.4b plots the logarithm of the expected first hitting-time $\ln E[\tau(i)]$. The lower and upper bounds from Theorem 4.4 are plotted in logarithm scale as dashed curves which almost coincide with the hitting-time plot. For better resolution we focused on a portion of the plot around the mode, the three curves becoming more distinguishable in Fig.4.4c. We can see that the mode $x^* = 333$ has $\pi(x^*) \approx 0.012$ and it is hit in $E[\tau_{x^*}] \approx 162$ times on average for q . This is much smaller than $n/2 = 500$ which would be the average time for exhaustive search. In comparison, for an uninformed (i.e uniform) proposal the result is $E[\tau_{x^*}] = 1000$. Thus, it becomes visible how a "good" proposal q can influence the speed of such a stochastic sampler.

Theorem 4.7. (Maciucă and Zhu, 2006) Let p and Q be the target probability and the proposal matrix respectively for a Metropolis-Hasting sampler. Let $M = \max_{i,j} Q_{ij}/p_j$ and $m = \min_{i,j} Q_{ij}/p_j$. We assume $m > 0$. Then for any initial distribution q , the expected f.h.ts are bounded by

$$p_i + \frac{1 - q_i}{M} \leq p_i E_q^Q[\tau(i)] \leq p_i + \frac{1 - q_i}{m}, \forall i.$$

Equality is attained if $Q_{ij} = p_j, \forall i, j$.

The proof is again in [127].

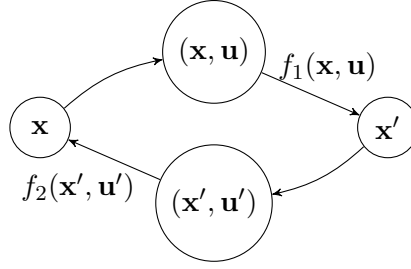
4.3 Reversible Jumps and Trans-Dimensional MCMC

There are many cases when one needs to sample a posterior probability that is defined over a union of spaces of different dimensions. For example, one could define Bayesian models for objects in images, with a variable number of parameters, and are interested in sampling from such models to estimate the most likely observation for a give image.

It was first proposed in Grenander and Miller 1994 [88] for image analysis and in Green 1995 [87] for Bayesian model selection.

4.3.1 Reversible Jumps

Let $\Omega = \cup_{i=1}^{\infty} \Omega_i$ be the solution space as a union of subspaces of different dimensions $\dim(\Omega_i) = d_i$ and π be a probability distribution defined on Ω . The reversible jumps are MCMC moves from one space Ω_i to another one Ω_j that satisfy the detailed balance equation with respect to π .



General case. The reversible jump move $q(\mathbf{x} \rightarrow \mathbf{x}')$ from $\mathbf{x} \in \Omega_i$ to $\mathbf{x}' \in \Omega_j$ is obtained by first sampling j from a probability $q(j|i, \mathbf{x})$ and sampling an auxiliary vector $\mathbf{u} \in \mathbb{R}^m$ (for some dimension m that needs to be specified) with p.d.f. $q(\mathbf{u}|\mathbf{x})$ and then obtaining \mathbf{x}' as a deterministic function $\mathbf{x}' = f_1(\mathbf{x}, \mathbf{u})$. A reverse move $q(\mathbf{x}' \rightarrow \mathbf{x})$ can be defined in a similar way, sampling i with probability $q(i|j, \mathbf{x}')$ and an auxiliary vector $\mathbf{u}' \in \mathbb{R}^{m'}$ from p.d.f. $q(\mathbf{u}'|\mathbf{x}')$. There must be a bijection $f : \Omega_i \times \mathbb{R}^m \rightarrow \Omega_j \times \mathbb{R}^{m'}$ such that $f(\mathbf{x}, \mathbf{u}) = (\mathbf{x}', \mathbf{u}')$, therefore the *dimension matching* condition $d_i + m = d_j + m'$ must be satisfied and $\frac{d\mathbf{x}'d\mathbf{u}'}{d\mathbf{x}d\mathbf{u}} = \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial(\mathbf{x}, \mathbf{u})}$. To satisfy detailed balance, the proposed move $q(\mathbf{x} \rightarrow \mathbf{x}')$ is accepted with probability:

$$\alpha(\mathbf{x} \rightarrow \mathbf{x}') = \min \left(1, \frac{q(j|i, \mathbf{x}')q(\mathbf{u}'|\mathbf{x}')\pi(\mathbf{x}')}{q(j|i, \mathbf{x})q(\mathbf{u}|\mathbf{x})\pi(\mathbf{x})} \left| \det \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial(\mathbf{x}, \mathbf{u})} \right| \right) \quad (4.4)$$

Expansion-contraction. A special case of reversible jumps are expansion-contraction moves where $\Omega_j = \Omega_i \times Z$. Starting from $\mathbf{x} \in \Omega_i$ one could choose $\mathbf{u} \in Z$ and f as the identity function, thus obtaining the expansion move $q(\mathbf{x} \rightarrow \mathbf{x}') = (\mathbf{x}, \mathbf{u})$. Starting from $\mathbf{x}' = (\mathbf{x}, \mathbf{u}) \in \Omega_j$ the contraction move just drops the \mathbf{u} , thus $q(\mathbf{x}' \rightarrow \mathbf{x}) = \mathbf{x}$. The acceptance probability for the expansion move is then

$$\alpha(\mathbf{x} \rightarrow \mathbf{x}') = \min \left(1, \frac{\pi(\mathbf{x}')}{\pi(\mathbf{x})q(\mathbf{u}|\mathbf{x})} \right) \quad (4.5)$$

and for the contraction is

$$\alpha(\mathbf{x}' \rightarrow \mathbf{x}) = \min \left(1, \frac{\pi(\mathbf{x})q(\mathbf{u}|\mathbf{x})}{\pi(\mathbf{x}')} \right) \quad (4.6)$$

4.3.2 Toy Example: 1D Range image segmentation

In Figure 4.5 is shown an example of a simulated 1D range image $I(x), x \in [0, 1]$. It is generated by adding Gaussian noise $N(0, \sigma^2)$ to the original surfaces I_o from Fig. 2b. I_o consists of an unknown number of k surfaces which could be either straight lines or circular arcs, separated by $k - 1$ change points,

$$0 = x_0 < x_1 < \dots < x_k = 1.$$

Let $l_i \in \{\text{line}, \text{circle}\}$ index the surface type in interval $[x_{i-1}, x_i)$ with parameters $\theta_i, i = 1, \dots, k$. For a straight line, $\theta = (s, \rho)$ represents the slope s and intercept ρ . For a circular arc, $\theta = (u, v, R)$ represents the center (u, v) and radius R . Thus, the 1D “world scene” is represented by a vector of random variables,

$$W = (k, \{x_i, i = 1, \dots, k - 1\}, \{(l_i, \theta_i), i = 1, \dots, k\}).$$

The surface I_o is fully determined by W with $I_o(x) = I_o(x, l_i, \theta_i), x \in [x_{i-1}, x_i), i = 1, \dots, k$.

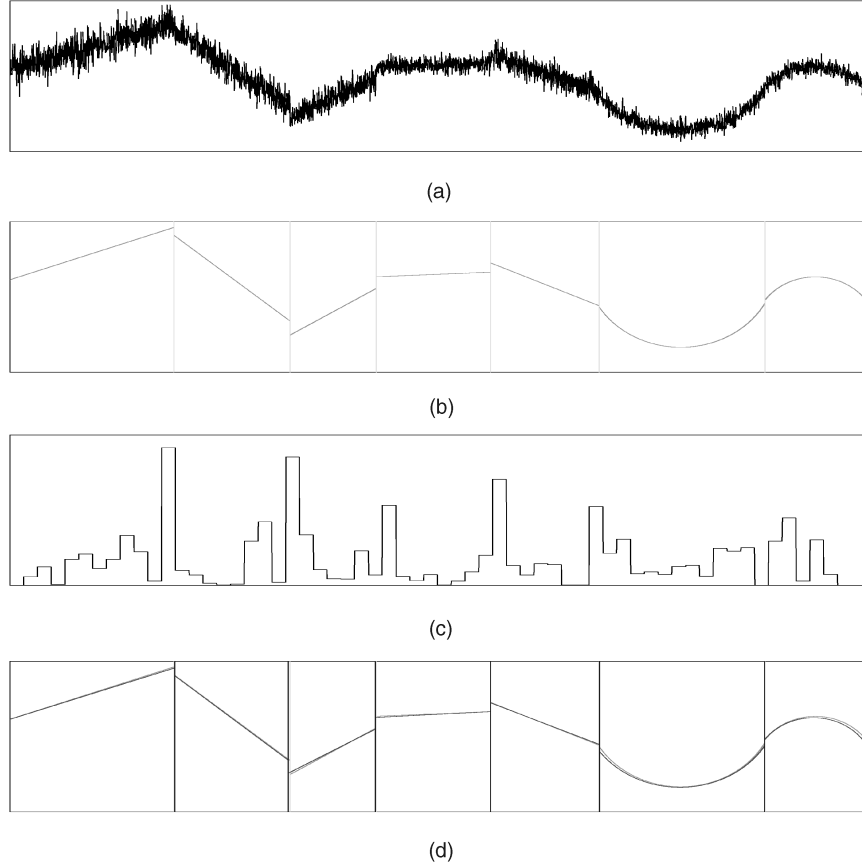


Figure 4.5: (a) A 1D range image $I(x), x \in [0, 1]$ (b) The true segmentation W_{th} . (c) Edginess measure $b(x), x \in [0, 1]$. A large value $b(x)$ indicates a high probability for x being a change point. (d) The best solution W^* (dark gray) found by the algorithm plotted against W_{th} (light gray).

By the standard Bayesian formulation, we have the posterior probability

$$p(W|I) \propto \exp \left\{ -\frac{1}{2\sigma^2} \sum_{i=1}^k \int_{x_{i-1}}^{x_i} (I(x) - I_o(x, l_i, \theta_i))^2 dx \right\} \cdot p(k) \prod_{i=1}^k p(l_i) p(\theta_i | l_i). \quad (4.7)$$

The first factor above is the likelihood and the rest are prior probabilities $p(k) \propto \exp(-\lambda_0 k)$ and $p(\theta_i | l_i) \propto \exp(-\lambda \# \theta_i)$ which penalizes the number of parameters $\# \theta_i$. $p(l_i)$ is a uniform probability on the lines and arcs. Thus, an energy function is defined,

$$E(W) = \frac{1}{2\sigma^2} \sum_{i=1}^k \int_{x_{i-1}}^{x_i} (I(x) - I_o(x, l_i, \theta_i))^2 dx + \lambda_0 k + \lambda \sum_{i=1}^k \# \theta_i. \quad (4.8)$$

The problem is that W does not have a fixed dimension. The probability $p(W|I)$ (or the energy $E(W)$) is thus distributed over a countable number of subspaces of varying dimension. The next subsection briefly introduces the jump diffusion process for exploring such solution spaces.

Jump-diffusion

The solution space is $\Omega = \cup_{n=1}^{\infty} \Omega_n$ where the subspace indexes $n = (k, l_1, \dots, l_k)$ contain the discrete variables of the model. To traverse the solution space $\Omega = \cup_{n=1}^{\infty} \Omega_n$, the algorithm needs two types of moves: reversible jumps between different subspaces and stochastic diffusions within each continuous subspace.

1. **Reversible jumps.** Let $W = (i, \mathbf{x})$, be the state of a Markov chain at time t , where $\mathbf{x} \in \Omega_i$ represents the continuous variables for the solution. In an infinitesimal time interval dt , the Markov chain jumps to a new state $W' = (j, \mathbf{x}')$ in another subspace $\Omega_j, j \neq i$. There are three types of moves: 1) switching a line to a circular arc or vice versa, 2) merging two adjacent intervals to a line or a circle, and 3) splitting an interval into two intervals (lines or circles). The jump is realized by a Metropolis move [139] that proposes to move from W to W' by a forward proposal probability $q(W'|W) = q(i \rightarrow j)q(\mathbf{x}'|j)$. The backward proposal probability is $q(W|W') = q(j \rightarrow i)q(\mathbf{x}|i)$. The forward proposal is accepted with probability

$$\alpha(W \rightarrow W') = \min \left(1, \frac{q(j \rightarrow i)q(\mathbf{x}|i)\pi(W')}{q(i \rightarrow j)q(\mathbf{x}'|j)\pi(W)} \right). \quad (4.9)$$

The dimension is matched in the above probability ratio.

2. **Stochastic diffusions.** Within each subspace Ω_n with $n = (k, l_1, \dots, l_k)$ fixed, the energy functional $E(\mathbf{x})$ is

$$E(\mathbf{x}) = E(x_1, \dots, x_{k-1}, \theta_1, \dots, \theta_k) = \frac{1}{2\sigma^2} \sum_{i=1}^k \int_{x_{i-1}}^{x_i} (I(x) - I_o(x, l_i, \theta_i))^2 dx + \text{const.}$$

We adopt a stochastic diffusion (or Langevin) equation to explore the subspace. The Langevin equations are the steepest descent PDE (partial differential equations) driven by Brownian motions $dB(t)$ with temperature T . Let $\mathbf{x}(t)$ denote the variables at time t ,

$$d\mathbf{x}(t) = -\frac{dE(\mathbf{x})}{d\mathbf{x}}dt + \sqrt{2T(t)}dw_t, \quad dw_t \sim N(0, (dt)^2). \quad (4.10)$$

For example, the motion equation of a change point x_i is

$$\frac{dx_i(t)}{dt} = \frac{1}{2\sigma^2} [(I(x) - I_o(x, l_{i-1}, \theta_{i-1}))^2 - (I(x) - I_o(x, l_i, \theta_i))^2] + \sqrt{2T(t)}N(0, 1).$$

This is the 1D version of the region competition equation [224]. The movement of the point x_i is driven by the fitness of data $I(x_i)$ to the surface models of the two adjacent intervals plus a Brownian motion. In practice, the Brownian motion is found to be useful in avoiding local pitfalls.

For computing the parameters $\theta_i, i = 1, \dots, k$ running the diffusion is more robust and often faster than fitting the best θ_i for each interval $[x_{i-1}, x_i)$ deterministically since the deterministic fit is an “overcommitment.” It is especially true when the current interval contains more than one objects.

It is well-known [77] that the continuous Langevin equations in (4.10) simulate Markov chains with stationary density $p(\mathbf{x}) \propto \exp(-E(\mathbf{x})/T)$. This is the posterior probability within subspace Ω_n at temperature T .

3. **The coordination of jumps and diffusions.** The continuous diffusions are interrupted by jumps at time instances $t_1 < t_2 < \dots < t_M \dots$ as Poisson events. In practice, the diffusion always runs at a discrete time step δt . Thus, the discrete waiting time τ_j between two consecutive jumps is

$$w = \frac{t_{j+1} - t_j}{\delta t} \sim p(w) = e^{-\tau} \frac{\tau^w}{w!},$$

where the expected waiting time $E(w) = \tau$ controls the frequency of jumps. Both jump and diffusion processes should follow an annealing scheme for lowering the temperature gradually.

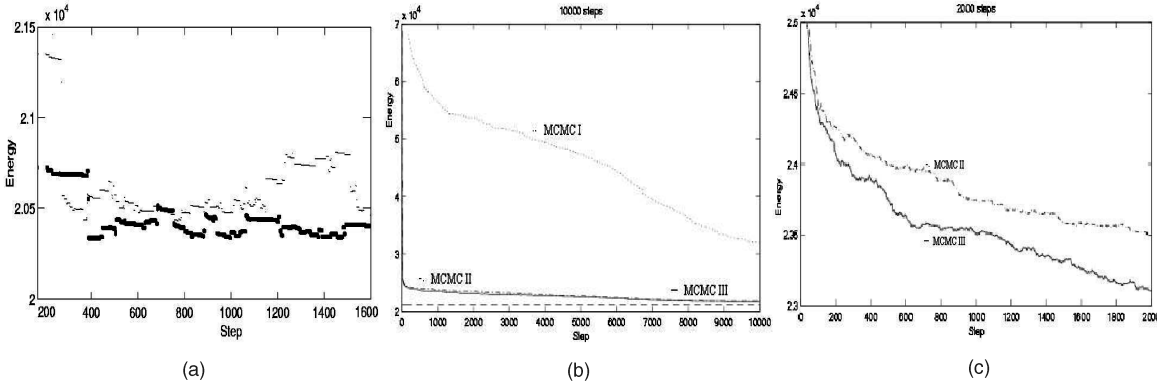


Figure 4.6: (a) Diffusion with jumps. Energy plots for two trials (MCMC II—thin curve and MCMC III—thick curve) of the jump-diffusion processes. Continuous energy changes in diffusion are interrupted by energy jumps. (b) Average energy plot. Comparison of the energy curves in the first 10,000 steps of three Markov chains MCMC I, II, and III averaged over 100 randomly generated signals. (c) Zoomed-in view of MCMC II and III for the first 2,000 steps. Note the energy scale is different from (b).

For illustration, Fig. 4.6a shows two trials (thin and thick curves respectively) of the jump-diffusion process running on the input 1D range data in Fig. 4.5a. The energy plots go up and down (i.e., the algorithm is not greedy) and the continuous energy curves (diffusion) are interrupted by jumps.

4. **Reversibility and global optimization.** From an engineering point of view, the most important property of the jump-diffusion process is that it simulates Markov chain to traverse the complex solution space. This property distinguishes it from greedy and local methods. In theory, this Markov chain samples from the posterior probability $p(W|I)$ over the solution space Ω [88]. With an annealing scheme, it can theoretically achieve the globally optimal solution with probability close to one. The reversibility of the jumps may not be a necessary condition; however, it is a useful tool for achieving irreducibility of the Markov chain in the complex solution space.
5. **The speed bottlenecks.** Conventional jump-diffusion designs are limited by their computing speed. However, this problem can be overcome by better design of the proposal probabilities as we shall show in the next section. We observed that the bottlenecks are in the jumps affected by the design of the proposal probabilities. In (4.9), a proposal probability $q(\mathbf{x}'|j)$ in interval $[x_{i-1}, x_i)$ can be divided into three cases: 1) switching to a new model with $\mathbf{x}' = \theta_i$,

2) merging to form a new interval x_{i-2}, x_i with type l and parameter \mathbf{x} , and 3) splitting to form two new intervals with models (l_a, θ_a) and (l_b, θ_b) respectively.

$$q(\mathbf{x}|m) = \begin{cases} q(\theta_i|l_i, [x_{i-1}, x_i]) & \text{switch } [x_{i-1}, x_i] \text{ to model } (l_i, \theta_i) \\ q(\theta|l, [x_{i-2}, x_i]) & \text{merge to a model } (l, \theta) \\ q(x|[x_{i-1}, x_i])q(\theta_a|l_a, [x_{i-1}, x])q(\theta_b|l_b, [x, x_i]) & \text{split } [x_{i-1}, x_i] \text{ into } (l_a, \theta_a) \text{ and } (l_b, \theta_b) \text{ at } x. \end{cases}$$

4.4 Application: Counting People

An application of the Metropolis-Hastings algorithm is presented in [72] for detecting and counting people in crowded scenes. The representation of the problem is in the framework of Marked Point Processes, where each person is represented as a marked point consisting of a spatial process representing the image location $p \in \mathbb{R}^2$ and a *marked process* denoted by $m = (w, h, \theta, j)$ representing the width, height, orientation and shape of the person, together forming the marked point $s = (p, (w, h, \theta, j))$.

4.4.1 Marked point process model

The model assumes the marked part depends on the spatial location from the marked process, therefore

$$\pi(s) = \pi(p)\pi(w, h, \theta, j|p)$$

for every marked point $s = (p, (w, h, \theta, j))$.

The prior for the point process $\pi(p)$ is a homogeneous Poisson point process, i.e. the total number of points follows a Poisson distribution and given the number of points, their location is uniform inside the region. A simulation of the prior model is shown in Figure 4.7.

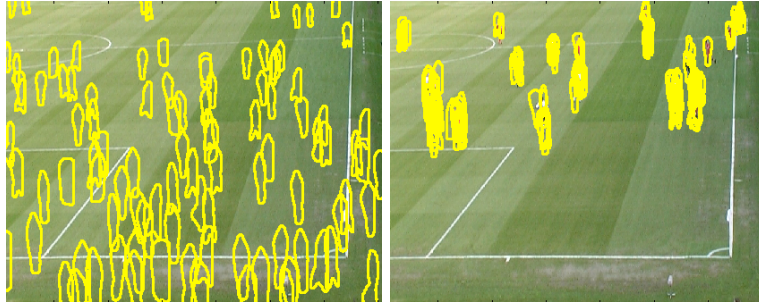


Figure 4.7: Samples from the Poisson point process prior $\pi(s)$.

The conditional mark process $\pi(w, h, \theta, j|p)$ is represented as independent Gaussians for the width, height and orientation, depending on the image location p , and a uniform distribution for the shape j from a set of possible shapes. The values of the spatially dependent means and variances are stored as look-up tables. The set of possible shapes are learned by Expectation Maximization as a mixture of Bernoulli templates from a set of manually segmented boxes.

The input image is processed so that a foreground *mask* data y is obtained where pixel i has $y_i = 1$ if the pixel is a foreground pixel and $y_i = 0$ for background.

Given the current point configuration s_1, \dots, s_n , a *label* image is constructed where the pixels are labeled foreground if any of the shapes corresponding to the n marked points covers it and zero

otherwise. In reality both the mask and label images have soft labels and contain values in the interval $[0, 1]$.

The likelihood is

$$\log \mathcal{L}(Y|X) = \sum (x_i \log y_i + (1 - x_i) \log(1 - y_i))$$

4.4.2 Inference by MCMC

Given an input image with a foreground mask y , the most likely marked point configuration is obtained by Maximum A Posteriori (MAP) estimation, i.e. maximization of the posterior probability function $\pi(s|y) = \pi(y|x(s))\pi(s)$.

This is achieved using three types of reversible moves:

- Birth proposal. A point with mark are proposed at uniform locations according to the foreground mask. The width, height and orientation are sampled from the respective Gaussians conditional on the point location. The type of the shape is chosen uniformly at random from the set of learned shape prototypes. The reverse of this mode is the death proposal.
- Death proposal. One point at random is removed.
- Update proposal. One marked point is selected at random and its position or mark parameters are modified. The position is modified as a random walk. The mark is modified by selecting one of the three parameters and sampling it from the conditional distribution given the current position, or selecting the shape type at random from the possible shapes.

The three types of moves are used with probability 0.4,0.2,0.4 respectively. About 500-3000 moves are needed for one image, starting from an empty configuration. More moves are needed for more crowded scenes.

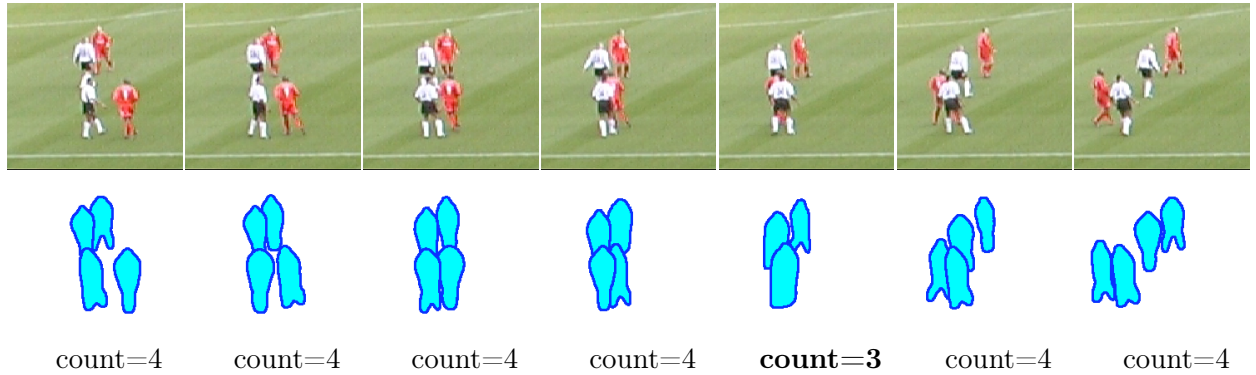


Figure 4.8: Results on seven frames of the VSPETS sequence. The counting is precise until there is significant overlap.

4.4.3 Results

The MCMC method was tested on two benchmark sequences with ground truth annotation: the EU CAVIAR dataset ¹ and the VSPETS soccer sequence ².

Examples of results are shown in Figures 4.8 and 4.9.

¹<http://homepages.inf.ed.ac.uk/rbf/CAVIAR/>

²<http://www.cvg.cs.rdg.ac.uk/VSPETS/vspets-db.html>



Figure 4.9: Results on images of the CAVIAR dataset.

4.5 Application: Furniture Arrangement

One application of the Metropolis-Hastings algorithm is furniture arrangement [214], illustrated in Figure 4.10. The approach consists of two stages: (1) the extraction of spatial, hierarchical, and pairwise relationships from positive examples and (2) the synthesis of novel furniture arrangements through optimization.



Figure 4.10: Left: Arbitrary initialization of the furniture layout. Middle and right: Two synthesized furniture arrangements optimized according to ergonomic criteria such as unobstructed accessibility and visibility.

1. Object representation. Optimizing furniture arrangement into a realistic and functional configuration relies on modeling various interaction factors, such as pairwise furniture relationships, spatial relationships with respect to the room, and other human factors.

Bounding surfaces: Each object in the scene is represented by a set of bounding surfaces. Figure 4.11 shows an example object (television) represented by a bounding box whose six surfaces are labeled 1 to 6. Apart from the top and bottom surfaces, there is a “back” surface for every object, which is the surface closest to a wall. Other surfaces are labeled as “non-back” surfaces. The back surface is used to define a reference plane for assigning other attributes.

Center and orientation: Figure 4.12(a) shows the key attributes of an object—center and orientation, denoted by (p_i, θ_i) , where p_i denotes the (x, y) coordinates and θ_i is the angle relative to the nearest wall (defined as the angle between the nearest wall and the back surface).

Accessible space: For each surface of the object, a corresponding accessible space is assigned (see Figure 4.12(b)). We define a_{ik} to be the center of coordinates of accessible space k of object i . The diagonal of the region is measured by ad_{ik} , which is used to measure how deep other objects can penetrate into the space during optimization. The size of the accessible space is set from available examples or given as input related to the size of a human body. If the space is very close to the wall in all the examples, the corresponding surface need not be accessible; otherwise, it is set to be the dimension of an average-sized adult if such a measurement is not given.

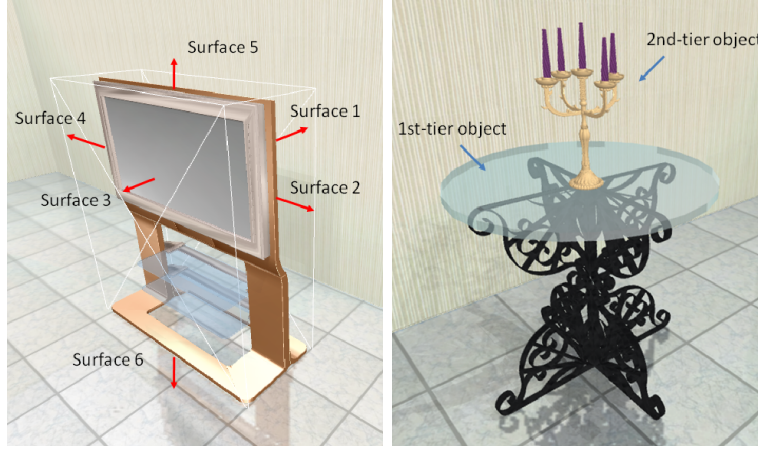


Figure 4.11: Left: A television, its bounding box, and six surfaces; Right: A candelabrum on a table; the table is a first-tier object and the candelabrum is a second-tier object.

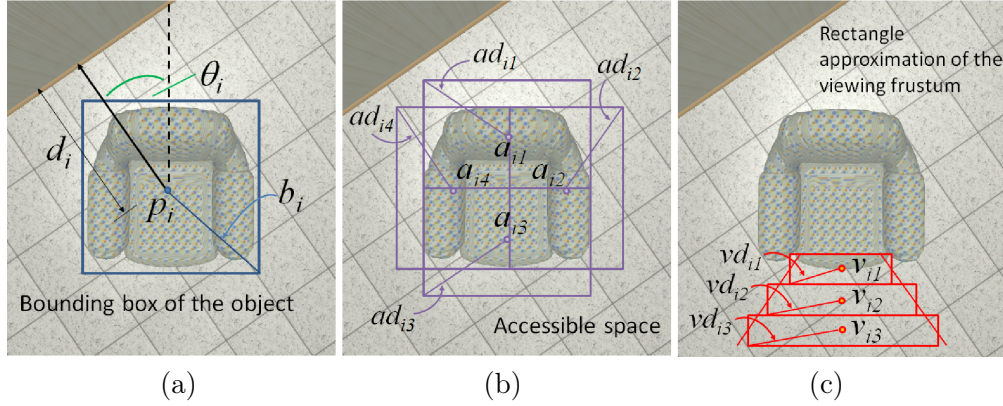


Figure 4.12: An example object i . (a) d_i is the distance of the object center p_i to the nearest wall, θ_i is the orientation of the object relative to the nearest wall, b_i is the diagonal of the bounding box. (b) The object has 4 accessible spaces centered at a_{i1} , a_{i2} , a_{i3} , and a_{i4} respectively. (c) A viewing frustum associated with the object is represented by 3 rectangles centered at v_{i1} , v_{i2} , v_{i3} , ad_{ik} and vd_{ik} are the corresponding diagonal lengths of the rectangles.

Viewing frustum: For some objects, such as the television and painting, the frontal surface must be visible. A viewing frustum is assigned to this particular surface, which for an object i , is approximated by a series of rectangles with center coordinates v_{ik} , where k is the rectangle index. vd_{ik} is the diagonal of the rectangle, which is useful in defining the penetration cost akin to that for the accessible space. Figure 4.12(c) provides an example.

Other attributes: Other attributes involved in the optimization process are the distance from p_i to its nearest wall is defined as d_i ; the diagonal b_i from p_i to the corner of the bounding box, as shown in Figure 4.12(a). Also recorded is the z -position z_i of the object.

2. Cost function. The goal of the optimization process is to minimize a cost function that characterizes realistic, functional furniture arrangements. Although it is often difficult to quantify the “realism” or “functionality” of a furniture arrangement, the following basic criteria should not be violated.

Accessibility: A furniture object must be accessible in order to be functional [35, 142]. To favor accessibility, the cost increases whenever any object moves into the accessible space of another object. Suppose object i overlaps with the accessible space k of object j , the accessibility cost is

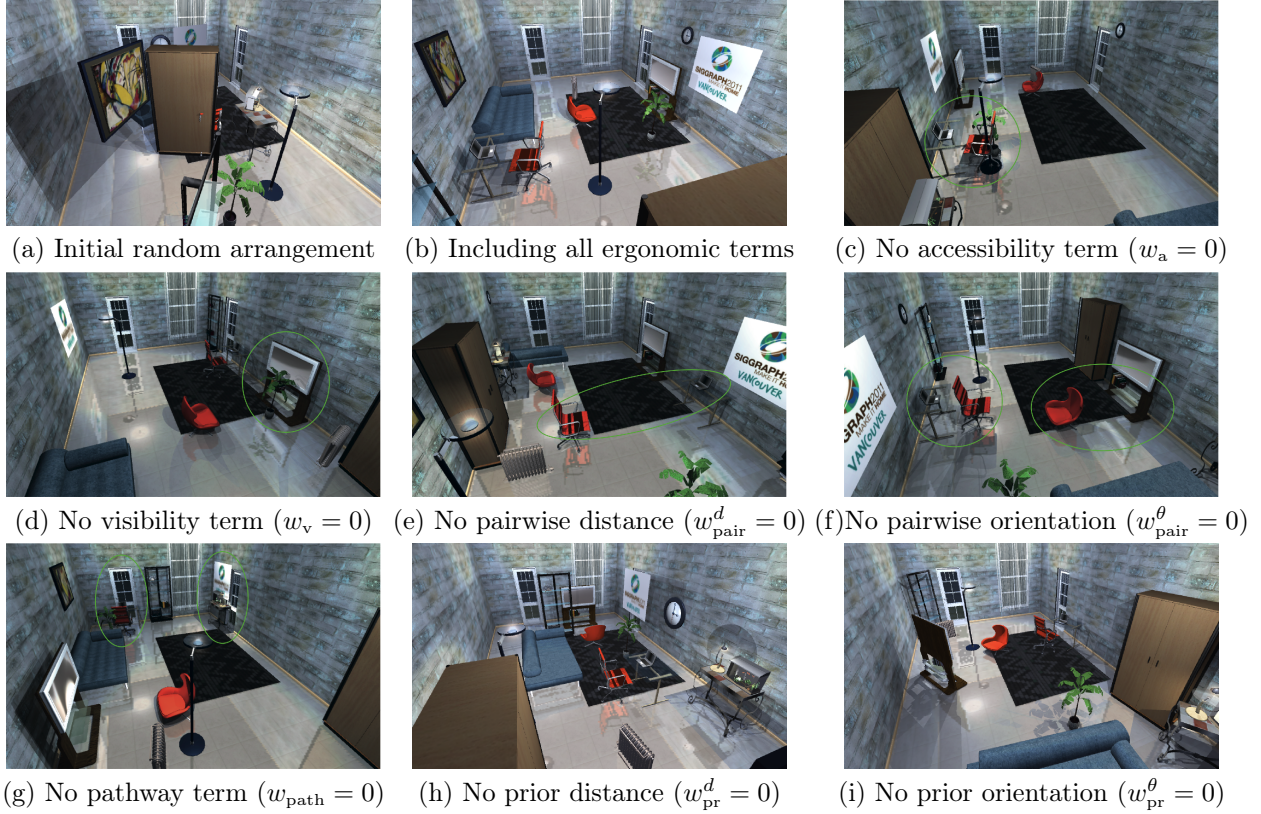


Figure 4.13: The effect on the automatic arrangement (b) of the furniture in (a) resulting from the omission of individual cost terms: Disregarding human ergonomics results in unrealistic synthesized arrangements that are not livable in several ways; e.g., (c) the furniture objects are colliding, (d) a potted plant is blocking the television and the armchair, (e) the work-chair is too far from the desk, (f) the armchair is facing away from the television, (g) the desk and work-chair are blocking the door, (h) furniture objects are too far from the wall, (i) objects are randomly oriented.

defined as

$$C_a(\phi) = \sum_i \sum_j \sum_k \max \left[0, 1 - \frac{\|p_i - a_{jk}\|}{b_i + ad_{jk}} \right]. \quad (4.11)$$

There are other cost terms for visibility, pathway connecting doors, pairwise constraints between certain furniture objects (eg. TV facing a sofa), and a prior to encourage configurations seen in the training examples. The effect of omitting individual cost terms is depicted in Figure 4.13.

3. Furniture arrangement optimization.

The search space of the problem is highly complex as objects are interdependent in the optimization process. The furniture positions and orientations depend on numerous factors, such as whether the object should be visible or accessible. It is very difficult to have a global optimization scheme or a closed-form solution that yields a unique optimum.

To tackle this problem, we resort to simulated annealing [101] with a Metropolis-Hastings state-search step [91, 139] to search for a good approximation to the global optimum. Note, however, that given a room, a set of furniture objects, and the prior spatial and hierarchical relationships, numerous acceptably-good configurations will be possible. This is the rationale for finding a good approximation in a reasonably short time, rather than searching exhaustively over the complex



Figure 4.14: Furniture arrangement optimization from a random initial configuration (left). As the optimization process proceeds, the furniture configuration is iteratively updated until it achieves an optimized final arrangement ϕ^* in 25,000 iterations (right).

search space in order to find the global optimum of the cost function.

To explore the space of possible arrangements effectively, the proposed moves $\phi \rightarrow \phi'$ involve both local adjustment, which modify the current arrangement, and a global reconfiguration step that swaps objects, thereby altering the arrangement significantly. There are three types of moves: translation and rotation, swapping objects and moving pathway control points. More details are given in [214].



Figure 4.15: Selected views of synthesized results. Top to bottom: Gallery, Resort, Restaurant.

With the aforementioned moves, given a floor-plan and a fixed number of furniture objects that define the solution space, the configuration of a furniture object (p_i, θ_i) has a positive probability to move to any other configuration (p'_i, θ'_i) . Given the annealing schedule, the solution space is explored more extensively with larger moves early in the optimization, and the furniture configuration is more finely tuned with smaller moves towards the end.

	Objects	Pairwise Relationships	Iterations	Time (sec)
<i>Living Room</i>	20	television & sofa	20000	22
<i>Bedroom</i>	24	television & armchair, desk & work chair	20000	48
<i>Restaurant</i>	54	chair & dish set, chair & table	25000	219
<i>Resort</i>	30	easel-stool, drum-chair, guitar-chair, couch-tea table	42000	126
<i>Factory</i>	51	work desk & chair, supervisor's desk & chair	42000	262
<i>Flower Shop</i>	64	none	22000	376
<i>Gallery</i>	35	chair & chair	18000	88

Table 4.1: Computation times are measured on a 3.33GHz Intel Xeon PC. Spatial and hierarchical relationships are extracted automatically from positive examples. Each pairwise relationship can be set by clicking the corresponding objects in the UI, whereupon the mean relative distance and angle are extracted from the examples for use as pairwise constraints.

Chapter 5

Gibbs Sampler and its Variants

The Gibbs sampler [75] is a MCMC algorithm for obtaining samples from distributions that are difficult to sample.

Usually the distributions are written in Gibbs form:

$$\pi(\mathbf{x}) = \frac{1}{Z} e^{-E(\mathbf{x})}, \quad \mathbf{x} = (x_1, \dots, x_d) \in \Omega.$$

Such distributions appear when solving constrained(hard, soft) satisfaction problems (e.g. image denoising) or in Bayesian inference. In such cases one would want to find the mode of the distribution, or certain distribution parameters such as mean, standard deviation, etc.

Before the Gibbs sampler, finding the mode was done by relaxation, using an algorithm similar to the one described in Figure 5.1.

Input: Energy function $E[\mathbf{x}]$, current state $\mathbf{x}^{(t)} = (x_1, \dots, x_d) \in \Omega$

Output: New state $\mathbf{x}^{(t+1)} \in \Omega$

1. Select a variable $i \in \{1, \dots, d\}$ at random

2. Compute:

$$u = \operatorname{argmin} \left(E[x_i = 1 | x_{-i}], \dots, E[x_i = L | x_{-i}] \right)$$

3. Set

$$\mathbf{x}_{-i}^{(t+1)} = \mathbf{x}_{-i}^{(t)}, x_i^{(t+1)} = u$$

Figure 5.1: Relaxation Algorithm

The problem with such a greedy algorithm is that it has no guarantees for finding the global optimum. In fact it very often gets stuck in local optima.

5.1 Gibbs Sampler

The goal of Gibbs Sampler is to sample a joint probability,

$$X = (x_1, x_2, \dots, x_d) \sim \pi(x_1, x_2, \dots, x_d)$$

It samples in each dimension according to the conditional probability,

$$x_i \sim \pi(x_i | \underbrace{x_{-i}}_{\text{fixed}}) = \frac{1}{Z} \exp(-E[x_i | x_{-i}]), \quad \forall i$$

where $\pi(x_i|x_{-i})$ is the conditional probability at a site (variable) i conditional of the other variables.

Suppose Ω is d -dimensional, and each dimension is discretized into L finite states, thus the total number of states is L^d . The procedure of Gibbs Sampler is shown in Figure 5.2.

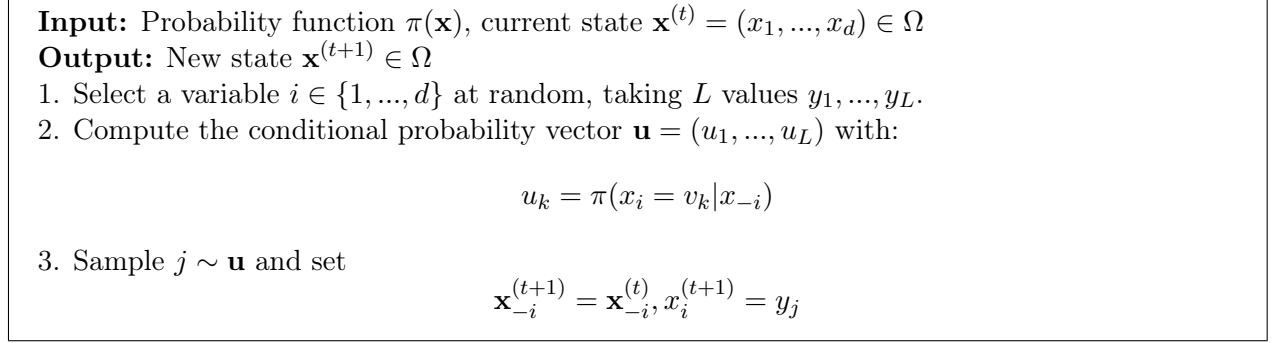


Figure 5.2: Gibbs Sampler

The order in which the variables are selected at step 1 above can be either randomized or follow a predefined scheme (e.g. $1, 2, \dots, d$).

Definition 5.1. A *sweep* of the Gibbs sampler is a sequential visit of all the sites (variables) once.

Although each the transition matrix K_i for one Gibbs step may not be irreducible and aperiodic, it is easy to show that the total transition matrix $K = K_1 \cdot K_2 \cdots K_d$ is irreducible and aperiodic after one sweep. Thus, the contraction coefficient $C(K) < 1$.

If $\mathbf{x}^{(t)} \sim \pi(\mathbf{x})$ at time t and $\mathbf{x}^{(t+1)} \sim \pi(\mathbf{x})$, then K has π as its invariant probability.

$$\begin{aligned}\mathbf{x}^{(t)} &= (x_1, \dots, x_i, x_{i+1}, \dots, x_d) \sim \pi(x) \\ \mathbf{x}^{(t+1)} &= (x_1, \dots, y_j, x_{i+1}, \dots, x_d)\end{aligned}$$

The only difference between the two is the x_i and y_j . However, we know that

$$\mathbf{x}^{(t+1)} \sim \pi(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) \cdot \pi(y_j | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) \implies \mathbf{x}^{(t+1)} \sim \pi(x)$$

In fact, one can show that the Gibbs sampler has a geometric rate of convergence:

$$\|\mu K^n - \pi\|_{\text{TV}} \leq \frac{1}{2}(1 - e^{N\Delta})^n \|\mu - \pi\|_{\text{TV}}$$

5.1.1 A major problem with the Gibbs sampler

The problem is illustrated in the following example.

Example 5.1. For a probability $\pi(x_1, x_2)$ whose probability mass is focused on a 1D line segment, as illustrated in Figure 5.3, sampling the two dimensional iteratively is obviously inefficient. i.e. the chain is “jagging”.

This is because the two variables are tightly coupled. It is best if we move along the direction of the line.

In general, problems arise when the probability is concentrated in a much lower dimensional *manifold* in the d -dimensional space. The Markov chain is not allowed to move in the normal directions (off the manifold) but only on the tangent directions.

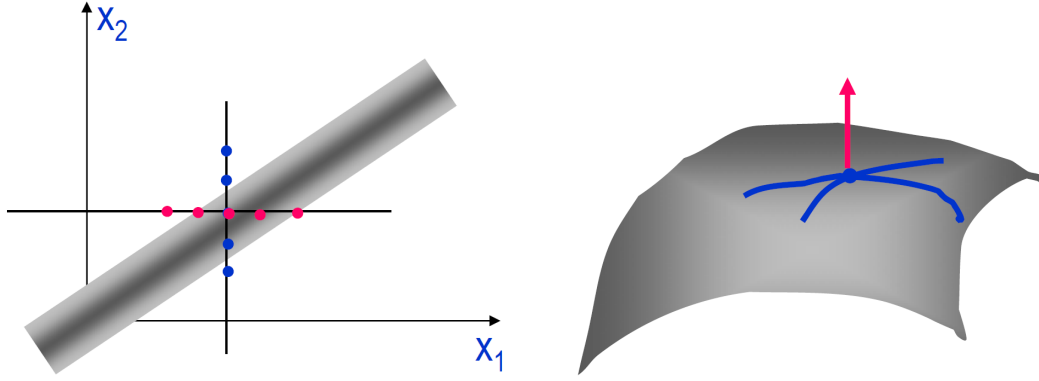


Figure 5.3: Left: The Gibbs sampler has a hard time sampling a probability with two tightly coupled variables, as explained in Example 5.1. Right: In general the Gibbs sampler has a hard time sampling data that is concentrated on a manifold.

As we know, Gibbs distributions are derived from constraints on the variable \mathbf{x} , and thus they are defined in some implicit manifold.

$$\Omega(H_0) = \{X : H_i(\mathbf{x}) = h_i, i = 1, 2, \dots, K\}, \quad H_0 = (h_1, h_2, \dots, h_K)$$

Examples of Gibbs distributions that could be hard to sample using the Gibbs sampler are Arkov Random Fields in general and the Ising/Potts model in particular.

Let $\mathbf{G} = \langle V, E \rangle$ be an adjacency graph, such as a lattice with the 4 nearest neighbor connections. Each vertex $v_i \in V$ has a state variable x_i with a finite number of labels (or colors), $x_i \in \{1, 2, \dots, L\}$. The total number of labels L is predefined.

Definition 5.2. Let $\mathbf{x} = (x_1, x_2, \dots, x_{|V|})$ denote the labeling of the graph, then the Ising/Potts model is a Markov random field,

$$\pi_{\text{PTS}}(\mathbf{x}) = \frac{1}{Z} \exp\left\{- \sum_{\langle s, t \rangle \in E} \beta_{st} \mathbf{1}(x_s \neq x_t)\right\}, \quad (5.1)$$

where $\mathbf{1}(x_s \neq x_t)$ is a Boolean function, equal to 1 if condition $x_s \neq x_t$ is satisfied, and is 0 otherwise. If the number of possible labels $L = 2$ it is called the Ising model, and for $L \geq 3$ it is the Potts model.

Usually we consider $\beta_{st} > 0$ for a ferro-magnetic system that prefers same colors for neighboring vertices. The Potts models and its extensions are used as *a priori* probabilities in many Bayesian inference tasks.

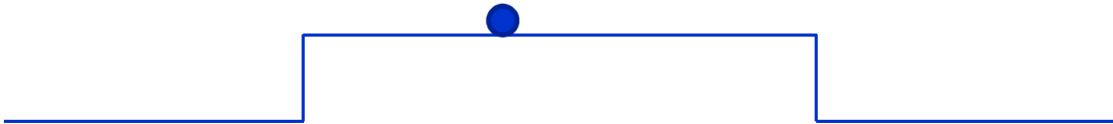


Figure 5.4: The Ising model has a flat energy landscape that is difficult to sample.

Example 5.2. For single site Gibbs sampler on the Ising model (5.1), the boundary spins are flipped with a $p = 1/2$ probability because the energy landscape is flat as illustrated in Figure 5.4. Flipping a string of length n will need on average $t \geq 1/p^n = 2^n$ steps! This is exponential waiting time.

5.2 Gibbs Sampler generalizations

This section presents some Gibbs sampler modifications and generalizations that alleviate some of the difficulties with using the Gibbs sampler for correlated variables highlighted in Section 5.1.1.

5.2.1 Hit-and-Run

This design selects a direction at random and samples in that direction.

Suppose the current state is $\mathbf{x}^{(t)}$.

- 1) Select a direction or axis \vec{e}_t .
- 2) Sample along the axis.

$$r \sim \pi(\mathbf{x}^{(t)} + r \cdot \vec{e}_t)$$

- 3) Update

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + r \cdot \vec{e}_t$$

The sampling along the axis will be a continuous Gibbs and implemented by Multi-Try Metropolis. However, there is still one problem with this design, which is how to select the sampling direction.

5.2.2 Generalized Gibbs Sampler

In fact, one may not have to move in straight lines. In more general cases, one may use a group of transformations for the possible moves, as long as the moves preserve the invariant probability.

Theorem (Liu and Wu, 1999 [125]). *Let $\Gamma = \{\gamma\}$ be a locally compact group that acts on the space Ω and each element multiplication is a possible move,*

$$\mathbf{x}^{(t)} \rightarrow \mathbf{x}^{(t+1)} = \gamma \cdot \mathbf{x}^{(t)}$$

If $\mathbf{x} \sim \pi$ and the element $\gamma \in \Gamma$ is chosen by

$$\gamma | \mathbf{x} \sim \pi(\gamma \cdot \mathbf{x}) | J_\gamma(\mathbf{x}) | H(d\gamma)$$

where $J_\gamma(x)$ is the Jacobian of the transformation $\mathbf{x} \rightarrow \gamma \cdot \mathbf{x}$ evaluated at \mathbf{x} and $H(d\gamma)$ is the left-invariant Haar measure,

$$H(\gamma \cdot B) = H(B), \quad \forall \gamma, B.$$

Then the new state follows the invariant probability

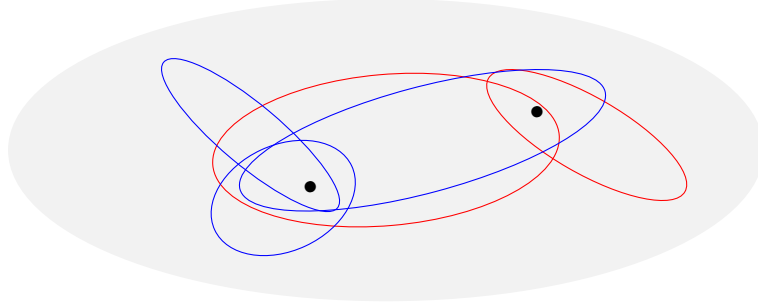
$$\mathbf{x}^{(t+1)} = \gamma \cdot \mathbf{x} \sim \pi$$

5.2.3 Generalized Hit-and-Run

Conceptually, it helps to generalize the hit-and-run idea to an *arbitrary partition* of the space, especially in finite state space. This is a concept by Persi Diaconis 2000.

Suppose a Markov Chain consists of many sub-chains, and the transition probability is a linear sum,

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^N \omega_i K_i(\mathbf{x}, \mathbf{y}), \quad \omega_i = p(i), \quad \sum_{i=1}^N \omega_i = 1$$



If each sub-kernel has the same invariant probability,

$$\sum_{\mathbf{x}} \pi(\mathbf{x}) K_i(\mathbf{x}, \mathbf{y}) = \pi(\mathbf{y}), \quad \forall \mathbf{y} \in \Omega$$

Then the whole Markov chain follows $\pi(\mathbf{x})$.

We denote the set of states connected to \mathbf{x} by the i -th type moves with kernel K_i as

$$\Omega_i(\mathbf{x}) = \{\mathbf{y} \in \Omega : K_i(\mathbf{x}, \mathbf{y}) > 0\}$$

Then \mathbf{x} is connected to the set

$$\Omega(\mathbf{x}) = \cup_{i=1}^N \Omega_i(\mathbf{x})$$

The key issues with this approach are:

1. How do we decide the sampling dimensions, directions, group transforms, and sets $\Omega_i(\mathbf{x})$ in a systematic and principled way?
2. How do we schedule the visiting order governed by $p(i)$? i.e. choosing the moving directions, groups, and sets

5.2.4 Sampling with auxiliary variables

We would like to sample $\mathbf{x} \sim \pi(\mathbf{x})$, which might be a distribution hard to sample due to correlations between variables. A systematic way to escape these correlations is to introduce auxiliary random variables:

$$\mathbf{x} \sim \pi(\mathbf{x}) \quad \rightarrow \quad (\mathbf{x}, y) \sim \pi^+(\mathbf{x}, y)$$

Examples for auxiliary variables y :

- T - temperature: Simulated Tempering [82]

- S – scale: Multi-grid sampling
- w – weight: Dynamic weighting
- b – bond: Cluster sampling, Swendsen-Wang [59, 180]
- u – energy level: Slice sampling [59]

5.2.5 Simulated Tempering

Let the *target probability* be

$$\pi(\mathbf{x}) = \frac{1}{Z} \exp\{-U(\mathbf{x})\}$$

Augment a variable I in $\{1, 2, \dots, L\}$ for L levels of temperature

$$1 = T_1 < T_2 < \dots < T_L$$

Sampling a joint probability, and keep the X 's with $I = 1$

$$(x, I) \sim \pi^+(x, I) = \frac{1}{Z^+} \exp\left\{-\frac{1}{T_I} U(\mathbf{x})\right\}$$

The sampler will move more freely in high temperature. But it is very difficult to cross between different temperature levels. Suppose we run Markov chains at the L levels in parallel. Define a joint probability for all chains

$$\pi^+(\mathbf{x}_1, \dots, \mathbf{x}_L) \propto \prod_{i=1}^L \exp\left\{-\frac{1}{T_i} U(\mathbf{x}_i)\right\}$$

Propose to permute two chains:

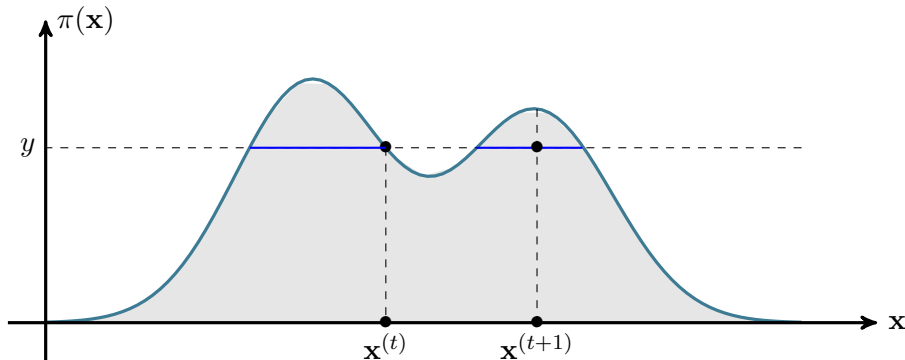
$$(\dots, \mathbf{x}_i, \dots, \mathbf{x}_j, \dots) \rightarrow (\dots, \mathbf{x}_j, \dots, \mathbf{x}_i, \dots)$$

Accept with Metropolis-Hastings

$$\alpha = \min\left(1, \exp\left\{\left(\frac{1}{T_j} - \frac{1}{T_i}\right)(U(\mathbf{x}_j) - U(\mathbf{x}_i))\right\}\right)$$

5.2.6 Slice Sampling

Suppose $\mathbf{x} \sim \pi(\mathbf{x})$ in a 1-D distribution. We introduce an auxiliary variable $y \in [0, 1]$ for the level of probability. Thus sampling $\pi(\mathbf{x})$ is equivalent to sampling uniformly from the shaded area in the (\mathbf{x}, y) space.



We satisfy the conditions of

$$\sum_y \pi^+(\mathbf{x}, y) = \pi(\mathbf{x})$$

but

$$\begin{cases} y \sim \pi^+(y|\mathbf{x}) = \text{unif}(0, \pi(\mathbf{x})) \leftarrow \text{easy to sample} \\ \mathbf{x} \sim \pi^+(\mathbf{x}|y) = \text{unif}(\overbrace{\{x; \pi(\mathbf{x}) \geq y\}}^{\text{level set}}) \leftarrow \text{hard to sample} \end{cases}$$

The slice $\{x; \pi(\mathbf{x}) \geq y\}$ usually contains multiple components bounded by the level set $\pi(bx) = y$ and is difficult to sample. This area is illustrated in Figure 5.5.

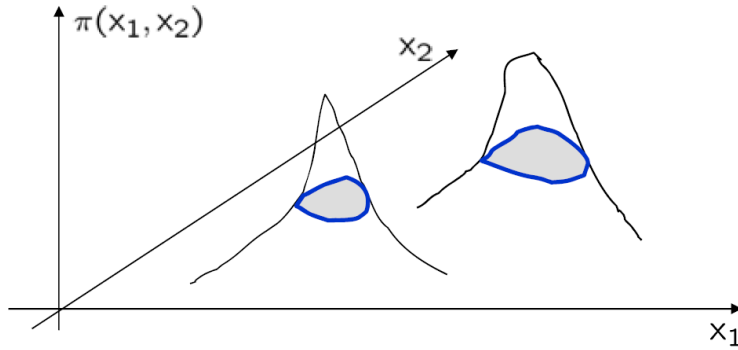


Figure 5.5: The slice $\{x; \pi(\mathbf{x}) \geq y\}$ usually contains multiple components and is difficult to sample.

5.2.7 Data Augmentation

The slice sampling method suggests two general conditions for auxiliary variables

$$\mathbf{x} \sim \pi(\mathbf{x}) \quad \rightarrow \quad (\mathbf{x}, y) \sim \pi^+(\mathbf{x}, y)$$

- 1) The marginal probability is

$$\sum_y \pi^+(\mathbf{x}, y) = \pi(\mathbf{x})$$

- 2) Both conditional probabilities are factorized and are easy to sample from

$$\begin{cases} \mathbf{x} \sim \pi^+(\mathbf{x}|y) \\ y \sim \pi^+(y|\mathbf{x}) \end{cases}$$

The intuitions for data augmentation is the following: Very often the probability is focused on separated modes (areas), and hopping between these modes is hard, since Markov chains usually move locally. Good auxiliary variables will:

- 1) help selecting moving directions/groups/sets (in generalized hit-and-run).
- 2) enlarge the search scopes.

5.2.8 Metropolized Gibbs Sampler

We go back to the generalized hit and run setup from section 5.2.3 where the kernel is made of a number of sub-kernels

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^N \omega_i K_i(\mathbf{x}, \mathbf{y}), \quad \omega_i = p(i), \quad \sum_{i=1}^N \omega_i = 1$$

with the same invariant probability, $\sum_{\mathbf{x}} \pi(\mathbf{x}) K_i(\mathbf{x}, \mathbf{y}) = \pi(\mathbf{y})$, $\forall \mathbf{y} \in \Omega$ and the set of states connected to \mathbf{x} by the i -th type moves is

$$\Omega_i(\mathbf{x}) = \{\mathbf{y} \in \Omega : K_i(\mathbf{x}, \mathbf{y}) > 0\}$$

Then \mathbf{x} is connected to the set

$$\Omega(\mathbf{x}) = \cup_{i=1}^N \Omega_i(\mathbf{x})$$

We know there are two general designs: Gibbs and Metropolis.

- 1) Gibbs Design, where we sample a probability in each set

$$y \sim [\pi]_i(\mathbf{y}), \quad [\pi]_i(\mathbf{y}) \sim \begin{cases} \pi(\mathbf{y}) & y \in \Omega_i(\mathbf{x}), \\ 0, & y \notin \Omega_i(\mathbf{x}) \end{cases}$$

In this way, the move is symmetric

$$\Omega_i(\mathbf{x}) = \Omega_i(\mathbf{y})$$

- 2) Metropolis Design. We know that in Metropolis design, we move arbitrary $\Omega_i(\mathbf{x})$, but we do not know the proposal distribution q .

$$q_i(\mathbf{x}, y) = \frac{\pi(\mathbf{y})}{\sum_{y' \in \Omega_i(\mathbf{x})} \pi(y')}, \quad \forall y' \in \Omega_i(\mathbf{x})$$

However, we will have to check

$$q_i(y, x) = \frac{\pi(\mathbf{x})}{\sum_{x' \in \Omega_i(\mathbf{y})} \pi(\mathbf{x}')}, \quad \forall x' \in \Omega_i(\mathbf{y})$$

The problem now is that it is no longer symmetric, $\Omega_i(\mathbf{x}) \neq \Omega_i(\mathbf{y})$. Although normalized, since the sets are different detailed balance equation might not be satisfied. To observe the detailed balance, we need a condition,

$$\mathbf{y} \in \Omega_i(\mathbf{x}) \quad \text{iff} \quad \mathbf{x} \in \Omega_i(\mathbf{y})$$

Acceptance probability,

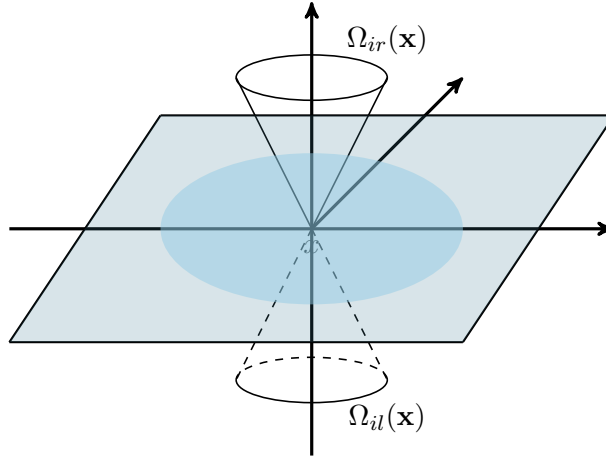
$$\alpha_i(\mathbf{x}, \mathbf{y}) = \min \left(1, \frac{q_i(\mathbf{y}, \mathbf{x}) \cdot \pi(\mathbf{y})}{q_i(\mathbf{x}, \mathbf{y}) \cdot \pi(\mathbf{x})} \right) = \min \left(1, \frac{\frac{\pi(\mathbf{x})}{\sum_{\mathbf{x}' \in \Omega_i(\mathbf{y})} \pi(\mathbf{x}') \cdot \pi(\mathbf{y})}}{\frac{\pi(\mathbf{y})}{\sum_{\mathbf{y}' \in \Omega_i(\mathbf{x})} \pi(\mathbf{y}') \cdot \pi(\mathbf{x})}} \right)$$

$$= \min \left(1, \frac{\overbrace{\sum_{\mathbf{y}' \in \Omega_i(\mathbf{x})} \pi(\mathbf{y}')}^{\text{total probability mass of } \Omega_i(\mathbf{x})}}{\underbrace{\sum_{\mathbf{x}' \in \Omega_i(\mathbf{y})} \pi(\mathbf{x}')}_{\text{total probability mass of } \Omega_i(\mathbf{y})}} \right)$$

The sub-kernels are designed in pairs,

$$K_i(\mathbf{x}, y) = \omega_{il} K_{il}(\mathbf{x}, y) + \omega_{ir} K_{ir}(\mathbf{x}, y)$$

and have their corresponding spaces $\Omega_{il}(\mathbf{x})$ and $\Omega_{ir}(\mathbf{x})$.



In this situation, the acceptance rate is

$$\alpha_i(\mathbf{x}, \mathbf{y}) = \min \left(1, \frac{\sum_{\mathbf{y}' \in \Omega_{il}(\mathbf{x})} \pi(\mathbf{y}')}{\sum_{\mathbf{x}' \in \Omega_{ir}(\mathbf{y})} \pi(\mathbf{x}')} \right), \quad \mathbf{y} \in \Omega_{il}(\mathbf{x})$$

If the sets are symmetric, *i.e.* $\Omega_{il}(\mathbf{x}) = \Omega_{ir}(\mathbf{y})$, then we will always have acceptance rate 1, *i.e.* $\alpha(\mathbf{x}, y) = 1$. If the sets are asymmetric, then we need the Metropolis acceptance step to “re-balance”.

One can improve the traditional Gibbs sampler by prohibiting the MC from staying in its current state in the conditional probability. Thus the sets are asymmetric and we need a Metropolis acceptance step to “re-balance”.

The diagonal elements in the *proposal matrix* are this way set to zero. This is a desirable property of MC design in order to make the MC “mix fast”.

$$q(\mathbf{x}, \mathbf{y}) = \frac{\pi(\mathbf{y})}{1 - \pi(\mathbf{x})}, \quad \mathbf{y} \in \Omega(\mathbf{x}), \mathbf{x} \notin \Omega(\mathbf{x})$$

where “1” represents the normalizing factor. Thus, the acceptance rate becomes

$$\alpha(\mathbf{x}, y) = \min \left(1, \frac{1 - \pi(\mathbf{x})}{1 - \pi(\mathbf{y})} \right)$$

Furthermore, it has been proven that

$$K_{\text{MGS}}(\mathbf{x}, \mathbf{y}) \geq K_{\text{Gibbs}}(\mathbf{x}, \mathbf{y}), \quad \forall \mathbf{x} \neq \mathbf{y} \in \Omega$$

5.3 Data association and data augmentation

There are many cases when accurate models $f(\mathbf{y}, \mathbf{h}|\theta)$ could be obtained when the observed data \mathbf{y} is augmented with some missing (hidden) data \mathbf{h} . For example a more accurate face model $f(\mathbf{y}, \mathbf{h}|\theta)$ (where $\theta \in 0, 1$ could represent face/nonface) can be obtained if the observed intensity image is augmented with a vector \mathbf{h} containing the face position, rotation, scale, 3D pose, and other variables (e.g. sun-glasses, beard, etc.).

Then the posterior distribution of the parameter θ conditional on the observed data is obtained by integrating out the hidden variables:

$$p(\theta|\mathbf{y}) = \int p(\theta|\mathbf{y}, \mathbf{h})p(\mathbf{h}|\mathbf{y})d\mathbf{h}. \quad (5.2)$$

If we could obtain samples $p(\mathbf{h}|\mathbf{y})$ then we could obtain a Monte Carlo approximation of $p(\theta|\mathbf{y})$ using Eq. (5.2) above.

Tanner and Wong [181] observed that we could use an initial approximation $f(\theta)$ of the target distribution $p(\theta|\mathbf{y})$ to obtain samples of the hidden variables $\mathbf{h}_1, \dots, \mathbf{h}_m$ from:

$$\tilde{p}(\mathbf{h}) = \int p(\mathbf{h}|\theta, \mathbf{y})f(\theta)d\theta$$

by first sampling $\theta_i \sim f(\theta)$ and then $\mathbf{h}_i \sim p(\mathbf{h}|\theta_i, \mathbf{y})$. These hidden samples are also called *multiple imputations*. We can use them to obtain a (hopefully) better approximation of the target distribution:

$$f(\theta) = \frac{1}{m} \sum_{i=1}^m p(\theta|\mathbf{y}, \mathbf{h}_i)$$

Thus the original data augmentation algorithm starts with a set of hidden values $\mathbf{h}_1^{(0)}, \dots, \mathbf{h}_m^{(0)}$ and proceeds as follows:

```

Initialize  $\mathbf{h}_1^{(0)}, \dots, \mathbf{h}_m^{(0)}$ 
for t=1 to  $N^{iter}$  do
  for i=1 to  $m$  do
    Pick  $k$  randomly from  $\{1, \dots, m\}$ 
    Sample  $\theta' \sim p(\theta|\mathbf{y}, \mathbf{h}_k^{(t-1)})$ 
    Sample  $\mathbf{h}_k^{(t)} \sim p(\mathbf{h}|\mathbf{y}, \theta')$ 
  end for
end for

```

Algorithm 2: The original Data Augmentation (DA) algorithm

An important observation to make is that the DA algorithm is equivalent to the version with $m = 1$, because one can trace back where each element of the current generation $\mathbf{h}_1^{(t)}, \dots, \mathbf{h}_m^{(t)}$ originates and it can be easily seen that when t is sufficiently large all samples from generation t

originate from a single element. Due to the purely random way the parents are chosen, there is no bias in the selection of the common ancestor.

Thus the DA algorithm is equivalent to a Gibbs sampler type of algorithm that alternates sampling the parameter θ with sampling the hidden variables \mathbf{h} :

```

Initialize  $\mathbf{h}$ 
for  $t=1$  to  $N^{iter}$  do
  Sample  $\theta' \sim p(\theta|\mathbf{y}, \mathbf{h}^{(t-1)})$ 
  Sample  $\mathbf{h}^{(t)} \sim p(\mathbf{h}|\mathbf{y}, \theta')$ 
end for

```

Algorithm 3: The simplified Data Augmentation (DA) algorithm

5.4 Julesz ensemble and MCMC sampling of texture

Before pursuing the “trichromacy” theory for texture, we will first review some important image features and statistics that have been used in texture modeling.

5.4.1 Image features and statistics

Let \mathbf{I} be an image defined on a finite lattice $\Lambda \subset \mathbb{Z}^2$. For each pixel $v = (x, y) \in \Lambda$, the intensity value at v is denoted by $\mathbf{I}(v) \in S$, with S being a finite interval on the real line or a finite set of quantized gray levels. We denote by $\Omega_\Lambda = S^{|\Lambda|}$ the space of all images on Λ .

In modeling homogeneous texture images, we start with exploring a finite set of statistics of some local image features. There are three major categories of image features studied in the literature.

The first category consists of k -gons proposed by Julesz. A k -gon is a polygon of k vertices indexed by $\alpha = (u_1, u_2, \dots, u_k)$, where $u_i = (\Delta x_i, \Delta y_i)$ is the displacement of the i -th vertex if we put the center of the k -gon at the origin. Δx_i and Δy_i must be integers. If we move this k -gon on the lattice, under some boundary conditions we collect a set of k -tuples

$$\{ (\mathbf{I}(v + u_1), \mathbf{I}(v + u_2), \dots, \mathbf{I}(v + u_k)) ; v \in \Lambda \}.$$

The k -gon statistic is the k -dimensional joint intensity histogram of these k -tuples, and it is also called the *co-occurrence matrix*.

The second type of features are the cliques in Markov random fields (MRF). Given a neighborhood system on the lattice Λ , a clique is a set of pixels that are neighbors of each other, so a clique is a special type of k -gon. Let $\alpha = (u_1, u_2, \dots, u_{k_\alpha})$ be the index for different types of cliques under a neighborhood system. According to the Hammersley-Clifford theorem [16], a Markov random field model has the Gibbs form

$$p(\mathbf{I}) = \frac{1}{Z} \exp\left\{-\sum_{\alpha} \sum_{v \in \Lambda} U_{\alpha}(\mathbf{I}(v + u_1), \mathbf{I}(v + u_2), \dots, \mathbf{I}(v + u_{k_\alpha}))\right\}$$

where Z is the normalization constant or the partition function and U_{α} are potential functions of k_{α} variables. The above Gibbs distribution can be derived from the maximum entropy principle under the constraints that $p(\mathbf{I})$ reproduces, on average, the co-occurrence matrices $\mathbf{h}^{(\alpha)}(b_1, \dots, b_{k_{\alpha}}; \mathbf{I}), \forall \alpha$. Therefore, the Gibbs model integrates all the co-occurrence matrices for the cliques into a single probability distribution. See Picard, Elfadel, and Pentland (1991) [160] for a related result. Like

k -gon statistics, this general MRF model also suffers from curse of dimensionality even for small cliques. The existing MRF texture models are much simplified in order to reduce the dimensionality of potential functions, such as in auto-binomial models [45], Gaussian MRF models [31] and ϕ -models [76].

The co-occurrence matrices (or joint intensity histograms) on the polygons and cliques have been proven inadequate for describing real world images and irrelevant to biologic vision systems. In the late 1980s, it was realized that real world imagery is better represented by spatial/frequency bases, such as Gabor filters [48], wavelet transforms [47], and filter pyramids. These filters are often called image *features*. Given a set of filters $\{F^{(\alpha)}, \alpha = 1, 2, \dots, K\}$, a sub-band image $\mathbf{I}^{(\alpha)} = F^{(\alpha)} * \mathbf{I}$ is computed for each filter $F^{(\alpha)}$.

Thus the third category of features extracts statistics on the sub-band images or pyramid instead of the intensity image. From a dimension reduction perspective, the filters characterize local texture features, as a result, very simple statistics of the sub-band images can capture information that would otherwise require k -gon or clique statistics of very high dimensions.

While Gabor filters are well grounded in biological vision [37], very little is known about how visual cortices pool statistics across images. There are four popular choices of statistics in the literature.

1. Moments of a single filter response, e.g. mean and variance of $\mathbf{I}^{(\alpha)}$.
2. Rectified functions that resemble the responses of "on/off" cells [5]:

$$\mathbf{h}^{(\alpha,+)}(\mathbf{I}) = \frac{1}{|\Lambda|} \sum_{v \in \Lambda} R^+(\mathbf{I}^{(\alpha)}(v)), \mathbf{h}^{(\alpha,-)}(\mathbf{I}) = \frac{1}{|\Lambda|} \sum_{v \in \Lambda} R^-(\mathbf{I}^{(\alpha)}(v)),$$

3. One bin of the empirical histogram of $\mathbf{I}^{(\alpha)}$.
4. One bin of the full joint histogram of $(\mathbf{I}^{(1)}, \dots, \mathbf{I}^{(K)})$.

As the second step to pursue the "trichromacy" theory of texture, in this section, we first propose a mathematical definition of texture—the Julesz ensemble, and then we study an algorithm for sampling images from the Julesz ensemble.

5.4.2 The Julesz ensemble - a mathematical definition of texture

Given a set of K statistics $\mathbf{h} = \{\mathbf{h}^{(\alpha)} : \alpha = 1, 2, \dots, K\}$ which have been normalized with respect to the size of the lattice $|\Lambda|$, an image \mathbf{I} is mapped into a point $\mathbf{h}(\mathbf{I}) = (\mathbf{h}^{(1)}(\mathbf{I}), \dots, \mathbf{h}^{(K)}(\mathbf{I}))$ in the *space of statistics*. Let

$$\Omega_{\Lambda}(\mathbf{h}_0) = \{\mathbf{I} : \mathbf{h}(\mathbf{I}) = \mathbf{h}_0\}$$

be the set of images sharing the same statistics \mathbf{h}_0 . Then the image space Ω_{Λ} is partitioned into equivalence classes

$$\Omega_{\Lambda} = \cup_{\mathbf{h}} \Omega_{\Lambda}(\mathbf{h}).$$

Due to intensity quantization in finite lattices, in practice one needs to relax the constraint on statistics and to define the image set as

$$\Omega_{\Lambda}(\mathcal{H}) = \{\mathbf{I} : \mathbf{h}(\mathbf{I}) \in \mathcal{H}\},$$

where \mathcal{H} is an open set around \mathbf{h}_0 .

$\Omega_{\Lambda}(\mathcal{H})$ implies a uniform distribution

$$q(\mathbf{I}; \mathcal{H}) = \begin{cases} \frac{1}{|\Omega_{\Lambda}(\mathcal{H})|} & \text{for } \mathbf{I} \in \Omega_{\Lambda}(\mathcal{H}), \\ 0 & \text{otherwise} \end{cases}$$

where $|\Omega_\Lambda(\mathcal{H})|$ is the volume of the set.

Definition Given a set of normalized statistics $\mathbf{h} = \{\mathbf{h}^{(\alpha)} : \alpha = 1, 2, \dots, K\}$, a Julesz ensemble $\Omega(\mathbf{h})$ is the limit of $\Omega_\Lambda(\mathcal{H})$ as $\Lambda \rightarrow \mathbb{Z}^2$ and $\mathcal{H} \rightarrow \{\mathbf{h}\}$ under some boundary conditions.

A Julesz ensemble $\Omega(\mathbf{h})$ is a mathematical idealization of $\Omega_\Lambda(\mathcal{H})$ on a large lattice with \mathcal{H} close to \mathbf{h} . As $\Lambda \rightarrow \mathbb{Z}^2$, it makes sense to let the normalized statistics $\mathcal{H} \rightarrow \{\mathbf{h}\}$. We assume $\Lambda \rightarrow \mathbb{Z}^2$ in the sense of van Hove [78], i.e., the ratio between the size of the boundary and the size of Λ goes to 0, $|\partial\Lambda|/|\Lambda| \rightarrow 0$. In engineering practice, we often consider a lattice big enough if $|\frac{\partial\Lambda}{\Lambda}|$ is very small, e.g. 1/15. Thus with a slight abuse of notation and also to avoid technicalities in dealing with limits, we consider a sufficiently large image (e.g. 256×256 pixels) as an infinite image in the rest of the paper. See the companion paper [208] for a more careful treatment.

A Julesz ensemble $\Omega(\mathbf{h})$ defines a texture pattern on \mathbb{Z}^2 , and it maps textures into the space of feature statistics \mathbf{h} . By analogy to color, as an electro-magnetic wave with wavelength $\lambda \in [400, 700]\text{nm}$ defines a unique visible color, a statistic value \mathbf{h} defines a texture pattern!¹ We shall study the relation between the Julesz ensemble and the mathematical models of texture in the next section.

A mathematical definition of texture could be different from a texture category in human texture perception. The latter has very coarse precision on the statistics \mathbf{h} and is often influenced by experience. For example, Julesz proposed that texture pairs which are not pre-attentively segmentable belong to the same category. Recently many groups have reported that texture pairs which are not pre-attentively segmentable by naive subjects become segmentable after practice [99]. This phenomenon is similar to color perception.

With the mathematical definition of texture, texture modeling is posed as an inverse problem. Suppose we are given a set of observed training images $\Omega_{\text{obs}} = \{\mathbf{I}_{\text{obs},1}, \mathbf{I}_{\text{obs},2}, \dots, \mathbf{I}_{\text{obs},M}\}$, which are sampled from an unknown Julesz ensemble $\Omega_* = \Omega(\mathbf{h}_*)$. The objective of texture modeling is to search for the statistics \mathbf{h}_* .

We first choose a set of K statistics from a dictionary B discussed in Section 5.4.1. We then compute the normalized statistics over the observed images $\mathbf{h}_{\text{obs}} = (\mathbf{h}_{\text{obs}}^{(1)}, \dots, \mathbf{h}_{\text{obs}}^{(K)})$, with

$$\mathbf{h}_{\text{obs}}^{(\alpha)} = \frac{1}{M} \sum_{i=1}^M \mathbf{h}^{(\alpha)}(\mathbf{I}_{\text{obs},i}), \quad \alpha = 1, 2, \dots, K. \quad (5.3)$$

Then we define an ensemble of texture images using \mathbf{h}_{obs} ,

$$\Omega_{K,\epsilon} = \{\mathbf{I} : D(\mathbf{h}^{(\alpha)}(\mathbf{I}), \mathbf{h}_{\text{obs}}^{(\alpha)}) \leq \epsilon, \quad \forall \alpha\}, \quad (5.4)$$

where D is some distance, such as the L_1 distance for histograms. If Λ is large enough to be considered infinite, we can set ϵ essentially at 0, and we denote the corresponding $\Omega_{K,\epsilon}$ as Ω_K . The ensemble Ω_K implies a uniform probability distribution $q(\mathbf{I}; \mathbf{h})$ over Ω_K , whose entropy is $\log |\Omega_K|$.

To search for the underlying Julesz ensemble Ω_* , one can adopt a pursuit strategy used by Zhu, Wu, and Mumford (1997) [222]. When $k = 0$, we have $\Omega_0 = \Omega_\Lambda$. Suppose at step k , a statistic \mathbf{h} is chosen, then at step $k + 1$ a statistic $\mathbf{h}^{(k+1)}$ is added to have $\mathbf{h}_+ = (\mathbf{h}, \mathbf{h}^{(k+1)})$. $\mathbf{h}^{(k+1)}$ is selected for the largest entropy decrease among all statistics in the dictionary B ,

$$\mathbf{h}^{(k+1)} = \arg \max_{\beta \in B} [\text{entropy}(q(\mathbf{I}; \mathbf{h})) - \text{entropy}(q(\mathbf{I}; \mathbf{h}_+))] = \arg \max_{\beta \in B} [\log |\Omega_k| - \log |\Omega_{k+1}|]. \quad (5.5)$$

¹We name this ensemble after Julesz to remember his pioneering work on texture. This does not necessarily mean that Julesz defined texture pattern with this mathematical formulation.

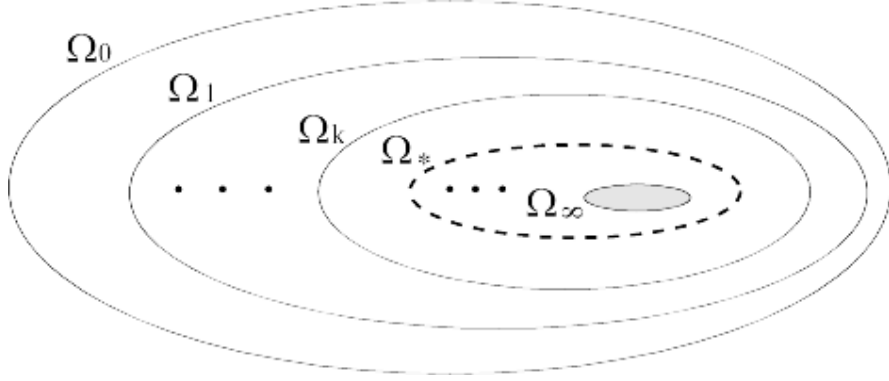


Figure 5.6: The volume (or entropy) of Julesz ensemble decreases monotonically with more statistical constraints added.

The decrease of entropy is called the *information gain* of $\mathbf{h}^{(k+1)}$.

As shown in figure 5.6, as more statistics are added, the entropy or volume of the Julesz ensemble decreases monotonically

$$\Omega_\Lambda = \Omega_0 \supseteq \Omega_1 \supseteq \cdots \supseteq \Omega_k \supseteq \cdots$$

Obviously introducing too many statistics will lead to an “over-fit”. In the limit of $k \rightarrow \infty$, Ω_∞ only includes the observed images in Ω_{obs} and their translated versions.

With the observed finite images, the choice of statistics \mathbf{h} and the Julesz ensemble $\Omega(\mathbf{h})$ is an issue of model complexity that has been extensively studied in the statistics literature. In the minimax entropy model [222, 223], an AIC criterion [2] is adopted for model selection. The intuitive idea of AIC is simple. With finite images, we should measure the fluctuation of the new statistics $\mathbf{h}^{(k+1)}$ over the training images in Ω_{obs} . Thus when a new statistic is added, it brings information as well as estimation error. The feature pursuit process should stop when the estimation error brought by $\mathbf{h}^{(k+1)}$ is larger than its information gain.

5.4.3 The Gibbs ensemble and ensemble equivalence

To make this paper self-contained, we briefly discuss in this section the Gibbs ensemble and the equivalence between the Julesz and Gibbs ensembles. A detailed study is referred to a companion paper [208].

Given a set of observed images Ω_{obs} and the statistics \mathbf{h}_{obs} , another line of research is to pursue probabilistic texture models, in particular the Gibbs distributions or Markov Random Field (MRF) models.

One general class of MRF model is the FRAME model studied by Zhu, Wu, and Mumford in 1997 [222, 223]. The FRAME model derived from the maximum entropy principle has the Gibbs form

$$p(\mathbf{I}; \boldsymbol{\beta}) = \frac{1}{Z(\boldsymbol{\beta})} \exp\left\{-\sum_{\alpha=1}^K \langle \beta^{(\alpha)}, \mathbf{h}^{(\alpha)}(\mathbf{I}) \rangle\right\} = \frac{1}{Z(\boldsymbol{\beta})} \exp\{\langle \boldsymbol{\beta}, \mathbf{h}(\mathbf{I}) \rangle\}. \quad (5.6)$$

The parameters $\boldsymbol{\beta} = (\beta^{(1)}, \beta^{(2)}, \dots, \beta^{(K)})$ are Lagrange multipliers. The values of $\boldsymbol{\beta}$ are determined so that $p(\mathbf{I}; \boldsymbol{\beta})$ reproduces the observed statistics,

$$E_{p(\mathbf{I}; \boldsymbol{\beta})}[\mathbf{h}^{(\alpha)}(\mathbf{I})] = \mathbf{h}_{\text{obs}}^{(\alpha)} \quad \alpha = 1, 2, \dots, K. \quad (5.7)$$

The selection of statistics is guided by a minimum entropy principle.

As the image lattice becomes large enough, the fluctuations of the normalized statistics diminish. Thus as $\Lambda \rightarrow \mathbb{Z}^2$, the FRAME model converges to a *limiting random field* in the absence of phase transition. The limiting random field essentially concentrates all its probability mass uniformly over a set of images which we call the *Gibbs ensemble*.²

In a companion paper [208], we prove that the Gibbs ensemble given by $p(\mathbf{I}; \boldsymbol{\beta})$ is equivalent to the Julesz ensemble specified by $q(\mathbf{I}; \mathbf{h}_{\text{obs}})$. The relationship between $\boldsymbol{\beta}$ and \mathbf{h}_{obs} is expressed in equation (5.7). Intuitively, $q(\mathbf{I}; \mathbf{h}_{\text{obs}})$ is defined by a “hard” constraint, while the Gibbs model $p(\mathbf{I}; \boldsymbol{\beta})$ is defined by a “soft” constraint. Both use the observed statistics \mathbf{h}_{obs} , and the model $p(\mathbf{I}; \boldsymbol{\beta})$ concentrates on the Julesz ensemble uniformly as the lattice Λ gets big enough.

The ensemble equivalence reveals two significant facts in texture modeling.

1. Given a set of statistics \mathbf{h} , we can synthesize typical texture images of the fitted FRAME model by sampling from the Julesz ensemble $\Omega(\mathbf{h})$ without learning the parameters $\boldsymbol{\beta}$ in the FRAME models [222], which is time consuming. Thus feature pursuit, model selection, and texture synthesis can be done effectively with the Julesz ensemble.
2. For images sampled from the Julesz ensemble, a local patch of the image given its environment follows the Gibbs distribution (or FRAME model) derived by the minimax entropy principle. Therefore, the Gibbs model $p(\mathbf{I}; \boldsymbol{\beta})$ provides a parametric form for the conditional distribution of $q(\mathbf{I}; \mathbf{h})$ on small image patches. $p(\mathbf{I}; \boldsymbol{\beta})$ should be used for tasks such as texture classification and segmentation.

The pursuit of Julesz ensembles can also be based on the minimax entropy principle. First, the definition of $\Omega(\mathbf{h})$ as the *maximum* set of images sharing statistics \mathbf{h} is equivalent to a maximum entropy principle. Second, the pursuit of statistics in equation (5.5) uses a minimum entropy principle. Therefore a unifying picture emerges for texture modeling under the minimax entropy theory.

5.4.4 Sampling the Julesz ensemble

Sampling the Julesz ensemble is by no means a trivial task! As $|\Omega_K|/|\Omega_\Lambda|$ is exponentially small, the Julesz ensemble has almost zero volume in the image space. Thus rejection sampling methods are inappropriate, and we resort to Markov chain Monte Carlo methods.

First, we define a function

$$G(\mathbf{I}) = \begin{cases} 0, & \text{if } D(\mathbf{h}^{(\alpha)}(\mathbf{I}), \mathbf{h}_{\text{obs}}^{(\alpha)}) \leq \epsilon, \quad \forall \alpha. \\ \sum_{\alpha=1}^K D(\mathbf{h}^{(\alpha)}(\mathbf{I}), \mathbf{h}_{\text{obs}}^{(\alpha)}), & \text{otherwise.} \end{cases}$$

Then the distribution

$$q(\mathbf{I}; \mathbf{h}, T) = \frac{1}{Z(T)} \exp\{-G(\mathbf{I})/T\} \quad (5.8)$$

goes to a Julesz ensemble Ω_K , as the temperature T goes to 0. The $q(\mathbf{I}; \mathbf{h}, T)$ can be sampled by the Gibbs sampler or other MCMC algorithms.

²In the computation of a feature statistic $\mathbf{h}(\mathbf{I})$, we need to define boundary conditions so that the filter responses in Λ are well defined. In case of phase transition, the limit of a Gibbs distribution is not unique, and it depends on the boundary conditions. However, the equivalence between Julesz ensemble and Gibbs ensemble holds even with phase transition. The study of phase transition is beyond the scope of this paper.

Algorithm I: Sampling the Julesz ensemble

Given texture images $\{\mathbf{I}_{\text{obs},i}, i = 1, 2, \dots, M\}$.
 Given K statistics (filters) $\{F^{(1)}, F^{(2)}, \dots, F^{(K)}\}$.
 Compute $\mathbf{h}_{\text{obs}} = \{\mathbf{h}_{\text{obs}}^{(\alpha)}, \alpha = 1, \dots, K\}$.
 Initialize a synthesized image \mathbf{I} (e.g. white noise).
 $T \leftarrow T_0$.
 Repeat
 Randomly pick a location $v \in \Lambda$,
 For $\mathbf{I}(v) \in S$ Do
 Calculate $q(\mathbf{I}(v) \mid \mathbf{I}(-v); \mathbf{h}, T)$.
 Randomly draw a new value of $\mathbf{I}(v)$ from $q(\mathbf{I}(v) \mid \mathbf{I}(-v); \mathbf{h}, T)$.
 Reduce T after each sweep.
 Record samples when $D(\mathbf{h}^{(\alpha)}(\mathbf{I}), \mathbf{h}_{\text{obs}}^{(\alpha)}) \leq \epsilon$ for $\alpha = 1, 2, \dots, K$.
 Until enough samples are collected.

In the above algorithm $q(\mathbf{I}(v) \mid \mathbf{I}(-v); \mathbf{h}, T)$ is the conditional probability of the pixel value $\mathbf{I}(v)$ with intensities for the rest of the lattice fixed. A sweep flips $|\Lambda|$ pixels in a random visiting scheme or to flip all pixels in a fixed visiting scheme.

Due to the equivalence between the Julesz ensemble and the Gibbs ensemble [208], the sampled images from $q(\mathbf{I}; \mathbf{h})$ and those from $p(\mathbf{I}; \boldsymbol{\beta})$ share the same statistics in that they produce not only the same statistics in \mathbf{h} , but also statistics extracted by any other filters, linear or nonlinear. It is worth emphasizing one key concept which has been misunderstood in some computer vision work: the Julesz ensemble is the set of “typical” images for the Gibbs model $p(\mathbf{I}; \boldsymbol{\beta})$, not the “most probable” images that minimize the Gibbs potential (or energy) in $p(\mathbf{I}; \boldsymbol{\beta})$.

One can use algorithm I for selecting statistics \mathbf{h} , as in [222]. That is, one can pursue new statistics by decreasing the entropy as measured in equation (5.5). An in-depth discussion is referred to [208].

5.4.5 Experiment: sampling the Julesz ensemble

In our first set of experiments, we select all of the 56 linear filters (Gabor filters at various scales and orientations and small Laplacian of Gaussian filters) used in [222]. The largest filter window size is 19×19 pixels. We choose \mathbf{h} to be the marginal histograms of these filters and sample the Julesz ensemble using algorithm I. Although only a small subset of filters are often necessary for each texture pattern, we use a common filter set in this section. We shall discuss statistics pursuit issues in the next section. It is almost impractical to learn a FRAME model integrating all these 56 filters in our previous work [222]; the computation is much easier using the simpler but equivalent model $q(\mathbf{I}; \mathbf{h})$.

We run the algorithm over a broad set of texture images collected from various sources. The results are displayed in figures 5.7. The left columns show the observed textures, and the right columns display the synthesized images whose sizes are 256×256 pixels. For these textures, the marginal statistics closely match (less than 1% error for each histogram) after about 20 to 100 sweeps, starting with a temperature $T_0 = 3$. Since the synthesized images are finite, the matching error ϵ cannot be infinitely small. In general, we set $\epsilon \propto \frac{1}{|\Lambda|}$.

These experiments demonstrate that Gabor filters and marginal histograms are sufficient for capturing a wide variety of homogeneous texture patterns. For example, the cloth pattern in the middle row of figure 5.7 has very regular structures, which are reproduced fairly well in the

synthesized texture image. This demonstrates that Gabor filters at various scales align up without using the joint histograms explicitly. The alignment or high order statistics are accounted for through the interactions of the filters.

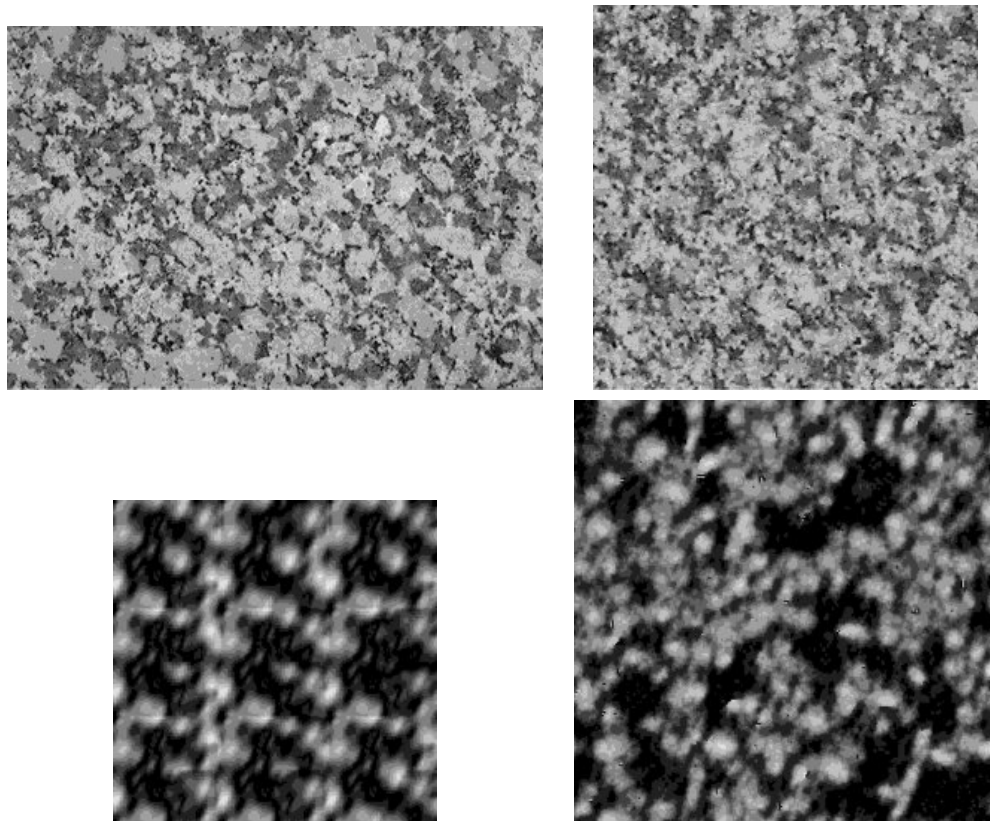


Figure 5.7: Left column: the observed texture images, right column: the synthesized texture images that share the exact histograms with the observed for 56 filters.

Our experiments reveal two problems.

The first problem is demonstrated in the failed example in figure 5.7. The observed texture patterns have large structures whose periods are longer than the biggest Gabor filter windows in our filter set. As a result, these periodic patterns are scrambled in the two synthesized images, while the basic texture features are well preserved.

The second problem is with the effectiveness of the Gibbs sampler. If we scale up the checker board image so that each square of the check board is 15×15 pixels in size, then we have to choose filters with large window sizes. It becomes infeasible to match the marginal statistics closely using the Gibbs sampler in algorithm I, since flipping one pixel at a time is inefficient for such large patterns. This suggests that we should search for more efficient sampling methods that can update large image patches. We believe that this problem would occur for other statistics matching methods, such as steepest descent [5, 69]. The inefficiency of the Gibbs sampler is also reflected in its slow mixing rate. After the first image is synthesized, it takes a long time for the algorithm to generate an image which is distinct from the first one. That is, the Markov chain moves very slowly in the Julesz ensemble.

Chapter 6

Cluster Sampling Methods

6.1 Introduction: Potts model and Swendsen-Wang

In this section, we review the Potts model and the Swendsen-Wang (SW) method.

Let $\mathbf{G} = \langle V, E \rangle$ be an adjacency graph, such as a lattice with 4 nearest neighbor connections. Each vertex $v_i \in V$ has a state variable x_i with a finite number of labels (or colors), $x_i \in \{1, 2, \dots, L\}$. The total number of labels L is predefined. Let $\mathbf{X} = (x_1, x_2, \dots, x_{|V|})$ denote the labeling of the graph, then the Ising/Potts model is a Markov random field,

$$\pi_{\text{PTS}}(\mathbf{X}) = \frac{1}{Z} \exp\left\{- \sum_{\langle s, t \rangle \in E} \beta_{st} \mathbf{1}(x_s \neq x_t)\right\}, \quad (6.1)$$

where $\mathbf{1}(x_s \neq x_t)$ is a Boolean function, equal to 1 if its condition $x_s \neq x_t$ is observed, and is 0 otherwise. If the number of possible labels $L = 2$ it is called the Ising model, and for $L \geq 3$ it is the Potts model. Usually we consider $\beta_{st} > 0$ for a ferro-magnetic system that prefers same colors for neighboring vertices. The Potts models and its extensions are used as *a priori* probabilities in many Bayesian inference tasks.

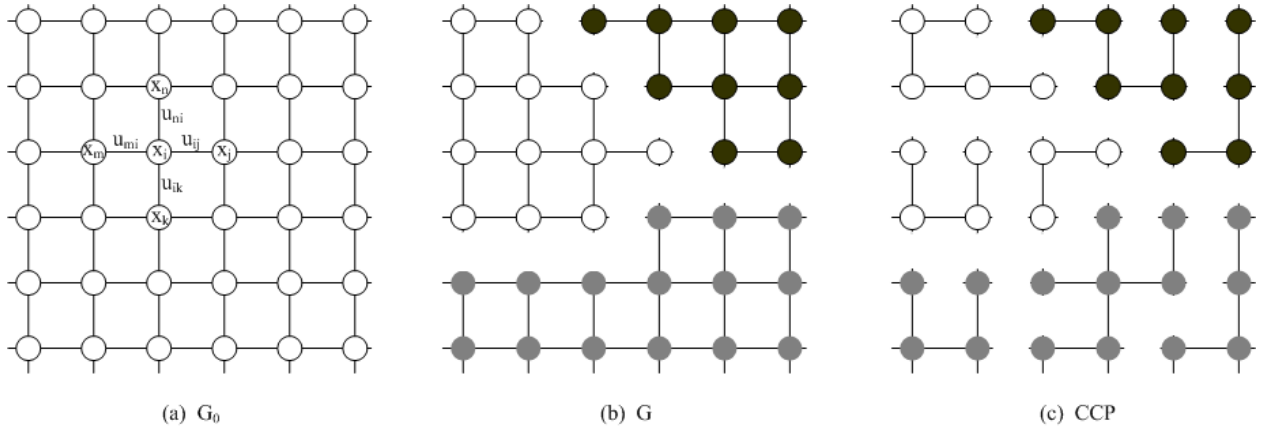


Figure 6.1: Illustrating the SW method. (a) An adjacency graph \mathbf{G} and each edge $e = \langle s, t \rangle$ is augmented with a binary variable $\mu_e \in \{1, 0\}$. (b) A labeling of the Graph \mathbf{G} where the edges connecting vertices of different colors are removed. (c). A number of connected component after turning off some edges in (b) probabilistically.

As Fig.6.1.(a) illustrates, the SW method introduces a set of auxiliary variables on the edges.

$$\mathbf{U} = \{\mu_e : \mu_e \in \{0, 1\}, \forall e \in E\}. \quad (6.2)$$

The edge e is disconnected (or turned off) if and only if $\mu_e = 0$. The binary variable μ_e follows a Bernoulli distribution conditional on x_s, x_t .

$$\mu_e | (x_s, x_t) \sim \text{Bernoulli}(q_e \mathbf{1}(x_s = x_t)), \quad \text{with } q_e = 1 - e^{-\beta_{st}}, \forall e \in E. \quad (6.3)$$

$\mu_e = 1$ with probability q_e if $x_s = x_t$, and $\mu_e = 0$ with probability 1 if $x_s \neq x_t$. The SW method iterates the following two steps:

1. The clustering step. Given the current state \mathbf{X} , it samples the auxiliary variables in \mathbf{U} according to eqn. (6.3). It first turns off each edges e according to μ_e . That is the edge $e = \langle s, t \rangle$ is turned off deterministically if $x_s \neq x_t$, as Fig.6.1.(b) shows.

$$E = E_{\text{on}}(\mathbf{X}) \cup E_{\text{off}}(\mathbf{X}). \quad (6.4)$$

The remaining edges are turned off the with probability $\rho_{st} = \exp(-\beta_{st})$. The edges e are divided into "on" and "off" sets respectively depending on $\mu_e = 1$ or 0 . Therefore we further divide the edge set $E_{\text{on}}(\mathbf{X})$,

$$E_{\text{on}}(\mathbf{X}) = E_{\text{on}}(\mathbf{U}, \mathbf{X}) \cup E_{\text{off}}(\mathbf{U}, \mathbf{X}). \quad (6.5)$$

The edges in $E_{\text{on}}(\mathbf{U}, \mathbf{X})$ form a number of connected components, as shown in Fig. 6.1.(c). We denote all the connected components given $E_{\text{on}}(\mathbf{U}, \mathbf{X})$ by,

$$\text{CP}(\mathbf{U}, \mathbf{X}) = \{\text{cp}_i : i = 1, 2, \dots, K, \text{ with } \cup_{i=1}^K \text{cp}_i = V\}. \quad (6.6)$$

Vertices in each connected component cp_i are guaranteed to have the same color. Intuitively, the strongly coupled sites have higher probability to be probabilistically grouped into a connected component. These connected components are "decoupled" for now.

2. The flipping step. It selects one connected component $V_o \in \text{CP}$ at random and assigns a common color ℓ to all vertices in V_o . The new label ℓ follows a uniform probability,

$$x_s = \ell \quad \forall s \in V_o, \quad \ell \sim \text{uniform}\{1, 2, \dots, L\}. \quad (6.7)$$

In this step, one may choose to repeat the random color flipping for one or all the connected components in $\text{CP}(\mathbf{U})$ independently, as they are decoupled given the edges in $E_{\text{on}}(\mathbf{U}, \mathbf{X})$. By doing so, all possible labelings of the graph are connected in one step, just like one sweep of the Gibbs sampler.

In one modified version by Wolff [205], one may choose a vertex $v \in V$ and grow a connected component following the Bernoulli trials on edges around v . This saves some computation in the clustering step, and thus bigger components have higher chance to be selected.

In Figure 6.2 are shown consecutive realizations obtained by running the SW algorithm on the Ising model for a lattice of size 256×256 for different values of the parameter $\beta_{ij} = \beta$. One can see that for small values of β the samples appear random and for $\beta = 1$ most nodes have the same label. There is a value β_0 of β around 0.8 or 0.9 at which there is a phase transition between the "random" phase and the "unicolor" phase. The value $1/\beta_0$ is called the *critical temperature*.

Using a path coupling technique, Cooper and Frieze [41] have showed that the mixing time τ (see eqn. (6.34)) is polynomial in the number of vertices N if each vertex in the graph G is connected

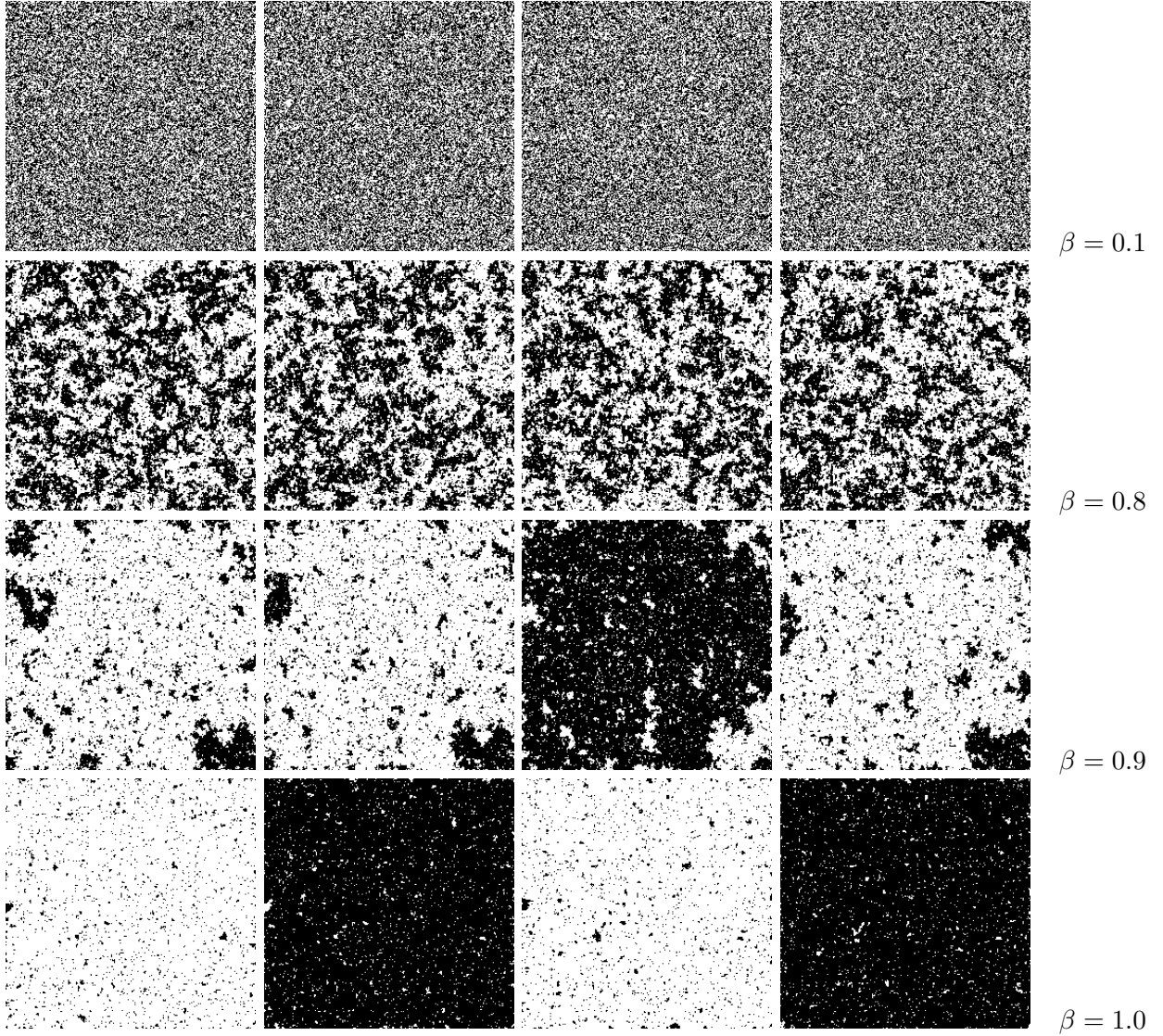


Figure 6.2: Consecutive samples obtained by the SW algorithm on the Ising model for different values of $\beta_{ij} = \beta$. From top to bottom: $\beta = 0.1, 0.8, 0.9, 1.0$ respectively.

to $O(1)$ neighbors, i.e. the connectivity of each vertex does not grow with the size of V . This is usually observed in computer vision problems, such as lattices or planar graphs. The mixing time becomes at worst case exponential when the graph G is fully connected [86]. Such cases usually don't occur in vision problems.

The Ising/Potts model $p(\mathbf{x})$ can be used as a prior for vision problems, in a Bayesian model $p(\mathbf{x}|I) \propto p(I|\mathbf{x})p(\mathbf{x})$ where the likelihood $p(I|\mathbf{x})$ measures how well the input image is explained by \mathbf{x} . However, the SW algorithm slows down in the presence of the likelihood, also known as an external field. This is because the clusters are created entirely based on the prior coefficients β_{ij} , ignoring the likelihood.

Higdon introduced an auxiliary variable method named partial decoupling [93] that takes into account the likelihood when growing the clusters. However, this approach is still limited to models with Ising/Potts priors.

Huber [95] developed a bounding chain method for Potts models (6.1) that can diagnose when the SW Markov chain has converged and thus obtain exact sampling or perfect sampling [166]. The number of steps for reaching exact sampling is in the order of $O(\log |E_o|)$ for temperatures that are far below or far above the critical temperature.

6.2 Interpretations of the SW Algorithm

There are three different interpretations of the SW algorithm, one as a Metropolis-Hasting algorithm, one as a data augmentation method with auxiliary variables and one as a slice sampling algorithm. For simplicity in this section we assume that we are working with a homogeneous Potts model with $\beta_{st} = \beta > 0, \forall s, t \in E$.

6.2.1 Interpretation 1: Metropolis-Hastings perspective

The SW algorithm can be interpreted as a Metropolis-Hastings step with acceptance probability 1.

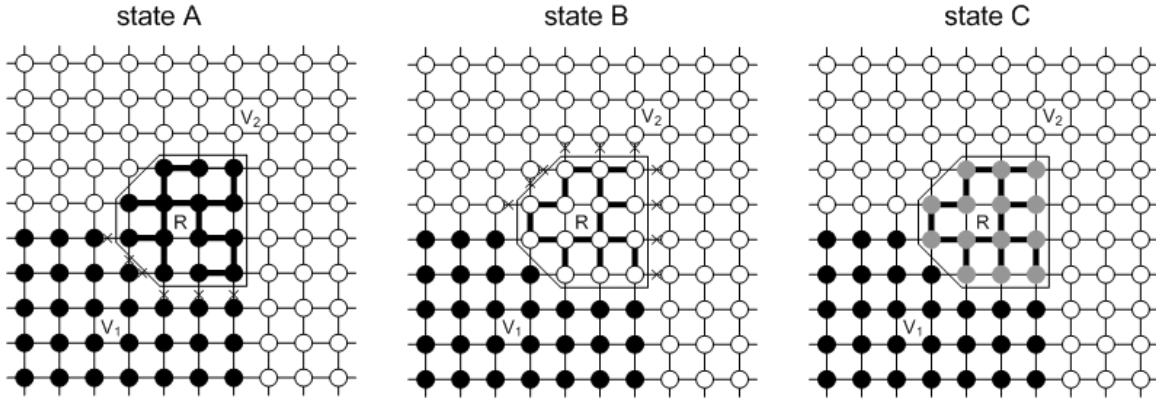


Figure 6.3: SW algorithm flips a patch of spins in one step for the Ising/Potts models.

Fig. 6.3 shows three partition states A , B and C which differ in the labels of the pixels in a connected component V_o (denoted by R in the figure). Suppose the current state is A in which V_o is connected to V_1 which are the remaining black vertices. The edges that are turned off probabilistically between V_o and V_1 is a cut

$$C_{01} = C(V_0, V_1) = \{e = \langle s, t \rangle : s \in V_0, t \in V_1\}.$$

The cut is illustrated by the crosses in Figure 6.3.

Obviously there are many ways to arrive at a connected component V_o through the random steps. But they must share a common cut $C(V_0, V_1)$.

Similarly if the Markov chain is currently at state B , it also has a chance to select a connected component V_o in white. We denote the remaining white vertices as V_2 , and the cut between V_0 and V_2 is

$$C_{02} = C(V_0, V_2) = \{e = \langle s, t \rangle : s \in V_0, t \in V_2\}.$$

So far, we have a pair of states A and B who are different in the labels of V_o . A Metropolis-Hastings method is used to realize a reversible move between them. Though it is difficult to compute the

proposal probabilities $Q(A \rightarrow B)$ and $Q(B \rightarrow A)$, one can compute their ratio easily through cancellation.

$$\frac{Q(A \rightarrow B)}{Q(B \rightarrow A)} = \frac{(1-q)^{|C_{01}|}}{(1-q)^{|C_{02}|}} = (1-q)^{|C_{01}| - |C_{02}|}. \quad (6.8)$$

In the above equation, $|C|$ is the cardinality of the set. In other words, the probabilities for selecting V_0 in states A and B are the same, except that the cuts are different. Remarkably the probability ratio for $\pi(A)/\pi(B)$ is also decided by the cuts through cancellation.

$$\frac{\pi(A)}{\pi(B)} = \frac{e^{-\beta|C_{02}|}}{e^{-\beta|C_{01}|}} = e^{\beta(|C_{01}| - |C_{02}|)} \quad (6.9)$$

The acceptance probability for the move from A to B is,

$$\alpha(A \rightarrow B) = \min(1, \frac{Q(B \rightarrow A)}{Q(A \rightarrow B)} \cdot \frac{\pi(B)}{\pi(A)}) = (\frac{e^{-\beta}}{1-q})^{|C_{01}| - |C_{02}|}. \quad (6.10)$$

By a smart choice of the edge probability

$$q = 1 - e^{-\beta},$$

then the proposal from A to B is always accepted with

$$\alpha(A \rightarrow B) = 1.$$

As $\beta \propto \frac{1}{T}$ is proportional to the inverse temperature, thus $q \rightarrow 1$ at low temperature and SW flips a large patch at a time. So SW algorithm can mix very fast at even critical temperature.

Proof of eq.(6.8). Here is the sketch of the proof. A generalization of this idea, with a complete proof will be given in Section 6.4.

Let $\mathbf{U}_A | (\mathbf{X} = A)$ and $\mathbf{U}_B | (\mathbf{X} = B)$ be the realizations of the auxiliary variables in states A and B respectively. Following the Bernoulli probabilities in the flipping step, which leads to two sets of connected components $\text{CP}(\mathbf{U}_A | \mathbf{X} = A)$ and $\text{CP}(\mathbf{U}_B | \mathbf{X} = B)$ respectively. We divide \mathbf{U}_A into two sets for the on and off edges respectively,

$$\mathbf{U}_A = \mathbf{U}_{A,\text{on}} \cup \mathbf{U}_{A,\text{off}}. \quad (6.11)$$

$$\mathbf{U}_{A,\text{on}} = \{\mu_e : \mu_e = 1\}, \quad \mathbf{U}_{A,\text{off}} = \{\mu_e : \mu_e = 0\}.$$

We are only interested in the \mathbf{U}_A 's (and thus $\text{CP}(\mathbf{U}_A | \mathbf{X} = A)$'s) which yield the connected component V_o . We collect all such \mathbf{U}_A given A in a set,

$$\Omega(V_o | A) = \{\mathbf{U}_A \text{ s.t. } V_o \in \text{CP}(\mathbf{U}_A | \mathbf{X} = A)\}. \quad (6.12)$$

In order for V_o to be a connected component in A , all edges between V_o and V_1 must be cut (turned off), otherwise V_o can not be a connected component. So, we denote the remaining "off" edges by ${}^-\mathbf{U}_{\text{off}}$,

$$\mathbf{U}_{A,\text{off}} = \mathcal{C}(V_o, V_1) \cup {}^-\mathbf{U}_{A,\text{off}}, \quad \forall \mathbf{U}_A \in \Omega(V_o | A). \quad (6.13)$$

Similarly, we collect all \mathbf{U}_B in state B which produce the connected component V_o ,

$$\Omega(V_o | B) = \{\mathbf{U}_B \text{ s.t. } V_o \in \text{CP}(\mathbf{U}_B | \mathbf{X} = B)\}. \quad (6.14)$$

In order for V_o to be a connected component in $\mathbf{U}_B|B$, the clustering step must cut all the edges between V_o and V_2 . Thus we have

$$\mathbf{U}_B = \mathbf{U}_{B,\text{on}} \cup \mathbf{U}_{B,\text{off}} \quad (6.15)$$

with

$$\mathbf{U}_{B,\text{off}} = \mathcal{C}(V_o, V_2) \cup {}^-\mathbf{U}_{B,\text{off}}, \quad \forall \mathbf{U}_B \in \Omega(V_o|B). \quad (6.16)$$

A key observation is that there is a one-to-one mapping between $\Omega(V_o|A)$ and $\Omega(V_o|B)$.

Proposition 6.1. *For any $\mathbf{U}_A \in \Omega(V_o|A)$, there exists one and only one $\mathbf{U}_B \in \Omega(V_o|B)$ such that*

$$\text{CP}(\mathbf{U}_A|\mathbf{X} = A) = \text{CP}(\mathbf{U}_B|\mathbf{X} = B) \quad (6.17)$$

and

$$\mathbf{U}_{A,\text{on}} = \mathbf{U}_{B,\text{on}}, \quad {}^-\mathbf{U}_{A,\text{off}} = {}^-\mathbf{U}_{B,\text{off}}. \quad (6.18)$$

That is, \mathbf{U}_A and \mathbf{U}_B differ only in the cuts $\mathcal{C}(V_o, V_1)$ and $\mathcal{C}(V_o, V_2)$.

Proof. Suppose that we choose $V_o \in \text{CP}(\mathbf{U}_A|\mathbf{X} = A)$ with uniform probability, then the probability for choosing V_o at state A is

$$q(V_o|A) = \sum_{\mathbf{U}_A \in \Omega(V_o|A)} \frac{1}{|\text{CP}(\mathbf{U}_A|\mathbf{X} = A)|} \prod_{e \in \mathbf{U}_{A,\text{on}}} q_e \prod_{e \in {}^-\mathbf{U}_{A,\text{off}}} (1 - q_e) \prod_{e \in \mathcal{C}(V_o, V_1)} (1 - q_e). \quad (6.19)$$

Similarly, the probability for choosing V_o at state B is

$$q(V_o|B) = \sum_{\mathbf{U}_B \in \Omega(V_o|B)} \frac{1}{|\text{CP}(\mathbf{U}_B|\mathbf{X} = B)|} \prod_{e \in \mathbf{U}_{B,\text{on}}} q_e \prod_{e \in {}^-\mathbf{U}_{B,\text{off}}} (1 - q_e) \prod_{e \in \mathcal{C}(V_o, V_2)} (1 - q_e). \quad (6.20)$$

Dividing eqn. (6.19) by eqn. (6.20), we obtain the ratio in eqn. (6.44) due to cancellation following the observations in Proposition 6.1. \square

In a special case when $\mathcal{C}(V_o, V_1) = \emptyset$, then $\prod_{e \in \mathcal{C}(V_o, V_1)} (1 - q_e) = 1$. Note that the proof holds for arbitrary design of q_e .

There is a slight complication, when there are two paths connecting the two states, as illustrated in Figure 6.4.

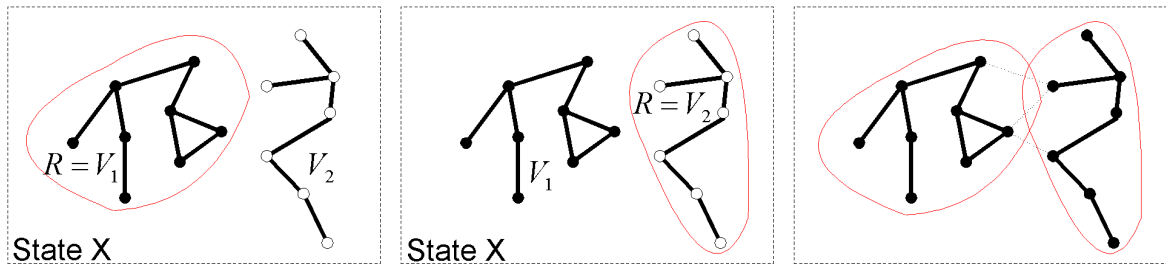


Figure 6.4: State A has two subgraphs V_1 and V_2 which are merged in state B .

Path 1. Choose $V_o = V_1$. In state A , choose new label $\ell = 2$, i.e. merge V_o to V_2 , and reversely in state B , choose new label $\ell = 1$, i.e. split V_o from V_2 .

Path 2. Choose $V_o = V_2$. In state A , choose new label $\ell = 1$, i.e. merge V_o to V_1 , and reversely in state B , choose $\ell = 2$, i.e. split V_o from V_1 . In such case, the proposal probability ratio is,

$$\frac{Q(B \rightarrow A)}{q(A \rightarrow B)} = \frac{Q(V_o = V_1|B)q(\mathbf{X}_{V_o} = 2|V_o, B) + q(V_o = V_2|B)q(\mathbf{X}_{V_o} = 1|V_o, B)}{q(V_o = V_1|A)q(\mathbf{X}_{V_o} = 1|V_o, A) + q(V_o = V_2|A)q(\mathbf{X}_{V_o} = 2|V_o, A)}. \quad (6.21)$$

In state A , the SW-cut $\mathcal{C}(V_o, V_\ell \setminus V_o) = \emptyset$ for both paths, and in state B the cut is $\mathcal{C}(V_1, V_2)$ for both paths. Following Prop. 6.7, the probability ratios for choosing $V_o = V_1$ and $V_o = V_2$ are equal,

$$\frac{q(V_o = V_1|A)}{q(V_o = V_1|B)} = \frac{1}{\prod_{e \in \mathcal{C}(V_1, V_2)} (1 - q_e)} = \frac{q(V_o = V_2|A)}{q(V_o = V_2|B)}. \quad (6.22)$$

Once V_o is selected, either $V_o = V_1$ or $V_o = V_2$, then the remaining partition for both A and B are the same, and is denoted by $\mathbf{X}_{V \setminus V_o} = \mathbf{X}_{V \setminus V_o}$. In proposing the new label of V_o , we easily observe that

$$\frac{q(\mathbf{X}_{V_o} = 2|V_o = V_1, B)}{q(\mathbf{X}_{V_o} = 1|V_o = V_2, A)} = \frac{q(\mathbf{X}_{V_o} = 1|V_o = V_2, B)}{q(\mathbf{X}_{V_o} = 2|V_o = V_1, A)}. \quad (6.23)$$

Then the acceptance rate remains 1.

6.2.2 Interpretation 2: data augmentation

The SW method described above is far from what was presented in the original paper [180]. Instead our description follows the interpretation by Edward and Sokal [59], who augmented the Potts model to a joint probability for both \mathbf{X} and \mathbf{U} ,

$$p_{\text{ES}}(\mathbf{X}, \mathbf{U}) = \frac{1}{Z} \prod_{e = \langle s, t \rangle \in E} [(1 - \rho)\mathbf{1}(\mu_e = 0) + \rho\mathbf{1}(\mu_e = 1) \cdot \mathbf{1}(x_s = x_t)] \quad (6.24)$$

$$= \frac{1}{Z} [(1 - \rho)^{|E_{\text{off}}(\mathbf{U})|} \cdot \rho^{E_{\text{on}}(\mathbf{U})}] \cdot \prod_{\langle s, t \rangle \in E_{\text{on}}(\mathbf{U})} \mathbf{1}(x_s = x_t). \quad (6.25)$$

The second factor $\prod_{\langle s, t \rangle \in E_{\text{on}}(\mathbf{U})} \mathbf{1}(x_s = x_t)$ is in fact a hard constraint on \mathbf{X} and \mathbf{U} . Let the space of \mathbf{X} be

$$\Omega = \{1, 2, \dots, L\}^{|V|}. \quad (6.26)$$

Under this hard constraint, the labeling \mathbf{X} is reduced to a quotient space $\frac{\Omega}{\text{CP}(\mathbf{U})}$ where each connected component must have the same label,

$$\prod_{\langle s, t \rangle \in E_{\text{on}}(\mathbf{U})} \mathbf{1}(x_s = x_t) = \mathbf{1}(\mathbf{X} \in \frac{\Omega}{\text{CP}(\mathbf{U})}). \quad (6.27)$$

The joint probability $p_{\text{ES}}(\mathbf{X}, \mathbf{U})$ observes two nice properties, both of which are easy to verify.

Proposition 6.2. *The Potts model is a marginal probability of the joint probability,*

$$\sum_{\mathbf{U}} p_{\text{ES}}(\mathbf{X}, \mathbf{U}) = \pi_{\text{PTS}}(\mathbf{X}). \quad (6.28)$$

The other marginal probability is the random cluster model π_{RCM} ,

$$\sum_{\mathbf{X}} p_{\text{ES}}(\mathbf{X}, \mathbf{U}) = \pi_{\text{RCM}}(\mathbf{U}) = \frac{1}{Z} (1 - \rho)^{|E_{\text{off}}(\mathbf{U})|} \cdot \rho^{E_{\text{on}}(\mathbf{U})} L^{|\text{CP}(\mathbf{U})|}. \quad (6.29)$$

Proposition 6.3. *The conditional probabilities of $p_{\text{ES}}(\mathbf{X}, \mathbf{U})$ are*

$$p_{\text{ES}}(\mathbf{U}|\mathbf{X}) = \prod_{\langle s, t \rangle \in E} p(\mu_e|x_s, x_t), \quad \text{with } p(\mu_e|x_s, x_t) = \text{Bernoulli}(\rho \mathbf{1}(x_s = x_t)), \quad (6.30)$$

$$p_{\text{ES}}(\mathbf{X}|\mathbf{U}) = \text{unif}\left[\frac{\Omega}{\text{CP}(\mathbf{U})}\right] = \left(\frac{1}{L}\right)^{|\text{CP}(\mathbf{U})|} \text{ for } \mathbf{X} \in \frac{\Omega}{\text{CP}(\mathbf{U})}; = 0 \text{ otherwise.} \quad (6.31)$$

Therefore the two SW steps can be viewed as sampling the two conditional probabilities.

1. Clustering step: $\mathbf{U} \sim p_{\text{ES}}(\mathbf{U}|\mathbf{X})$, i.e. $\mu_e|(x_s, x_t) \sim \text{Bernoulli}(\rho \mathbf{1}(x_s = x_t))$.
2. Flipping step: $\mathbf{X} \sim p_{\text{ES}}(\mathbf{X}|\mathbf{U})$, i.e. $\mathbf{X}(\text{cp}_i) \sim \text{Unif}\{1, 2, \dots, L\}$, $\forall \text{cp}_i \in \text{CP}(\mathbf{U})$.

As $(\mathbf{X}, \mathbf{U}) \sim p_{\text{ES}}(\mathbf{X}, \mathbf{U})$, discarding the auxiliary variables \mathbf{U} , we have \mathbf{X} following the marginal of $p_{\text{ES}}(\mathbf{X}, \mathbf{U})$. The goal is achieved,

$$\mathbf{X} \sim \pi_{\text{PTS}}(\mathbf{X}). \quad (6.32)$$

The beauty of this data augmentation method (Tanner and Wong [181]) is that the labels of the connected components are completely decoupled (independent) given the auxiliary variables. As $\rho = 1 - e^{-\beta}$, it tends to choose smaller clusters if the temperature ($T \propto \frac{1}{\beta}$) in the Potts model is high, and in low temperature it chooses large clusters. So it can overcome the coupling problem with single site Gibbs sampler.

6.3 Some theoretical results

Let the Markov chain have kernel \mathcal{K} and initial state \mathbf{X}_o , in t steps the Markov chain state follows probability $p_t = \delta(\mathbf{X} - \mathbf{X}_o)\mathcal{K}^t$ where $\delta(\mathbf{X} - \mathbf{X}_o)$ (for $\delta(\mathbf{X} - \mathbf{X}_o) = 1$ for $\mathbf{X} = \mathbf{X}_o$ and 0 otherwise) is the initial probability. The convergence of the Markov chain is often measured by the total variation

$$\|p_t - \pi\|_{\text{TV}} = \frac{1}{2} \sum_{\mathbf{X}} |p_t(\mathbf{X}) - \pi(\mathbf{X})|. \quad (6.33)$$

The mixing time of the Markov chain is defined by

$$\tau = \max_{\mathbf{X}_o} \min\{t : \|p_t - \pi\|_{\text{TV}} \leq \epsilon\}. \quad (6.34)$$

τ is a function of ϵ and the graph complexity $M = |\mathbf{G}_o|$ in terms of the number of vertices and connectivity. The Markov chain is said to mix rapidly if $\tau(M)$ is polynomial or logarithmic.

Empirically, the SW method is found to mix rapidly. Recently some analytic results on its performance have surfaced. Cooper and Frieze [41] proved using a path coupling technique that SW mixes rapidly on sparsely connected graphs.

Theorem 6.4. (Cooper and Frieze 1999) *Let $n = |V|$ and Δ be the maximum number of edges at any single vertex, and L the number of colors in Potts model. If \mathbf{G} is a tree, then the SW mixing time is $O(n)$ for any β and L . If $\Delta = O(1)$, then there exists $\rho_o = \rho(\Delta)$ such that if $\rho \leq \rho_o$ (i.e. higher than a certain temperature), then SW has polynomial mixing time for all L .*

A negative case was constructed by Gore and Jerrum [86] on complete graph.

Theorem 6.5. (Gore and Jerrum 1997) *If \mathbf{G} is a complete graph and $L > 2$, then for $\beta = \frac{2(L-1)\ln(L-1)}{n(L-2)}$, the SW does not mix rapidly.*

In the image analysis applications, our graph often observes the Copper-Frieze condition and the graph is far from being complete.

Most recently an exact sampling technique was developed for SW on Potts by Huber [95] for very high or very low temperatures. It designs a bounding chain which assumes that each vertex $s \in V$ has a set of colors S_s initialized with the full set $|S_s| = L$, $\forall s$. The Bernoulli probability for the auxiliary variables μ_e is changed to

$$\mathbf{U}^{\text{bd}} = \{\mu_e^{\text{bd}} : \mu_e^{\text{bd}} \in \{0, 1\}, \mu_e \sim \text{Bernoulli}(\rho \mathbf{1}(S_s \cap S_t \neq \emptyset))\}. \quad (6.35)$$

Thus \mathbf{U}^{bd} has more edges than \mathbf{U} in the original SW chain, i.e. $\mathbf{U} \subset \mathbf{U}^{\text{bd}}$. When \mathbf{U}^{bd} collapses to \mathbf{U} , then all SW chains starting with arbitrary initial states have collapsed into the current single chain. Thus it must have converged (exact sampling). The step for collapsing is called the "coupling time".

Theorem 6.6. (Huber 2002) *Let $n = |V|$ and $m = |E|$, at high temperature, $\rho < \frac{1}{2(\Delta-1)}$, the bounding chain couples completely by time $O(\ln(2m))$ with probability at least $1/2$. At lower temperature, $\rho \geq 1 - \frac{1}{mL}$, then the coupling time is $O((mL)^2)$ with probability at least $1/2$.*

In fact the Huber bound is not very tight as one may expect. Fig. 6.5(a) plots the results on a 5×5 lattice with torus boundary condition on the Ising model for the empirical coupling time against $\rho = 1 - e^{-\beta}$. The coupling time is large near the critical temperature (didn't plot). The Huber bound for the high temperature starts with $\rho_o = 0.16$ and is plotted by the short curve. The bound for the low temperature starts with $\rho_o > 0.99$ which is not visible. Fig. 6.5(b) plots the coupling time at $\rho = 0.15$ against the graph size $m = |E|$ and the Huber bound.

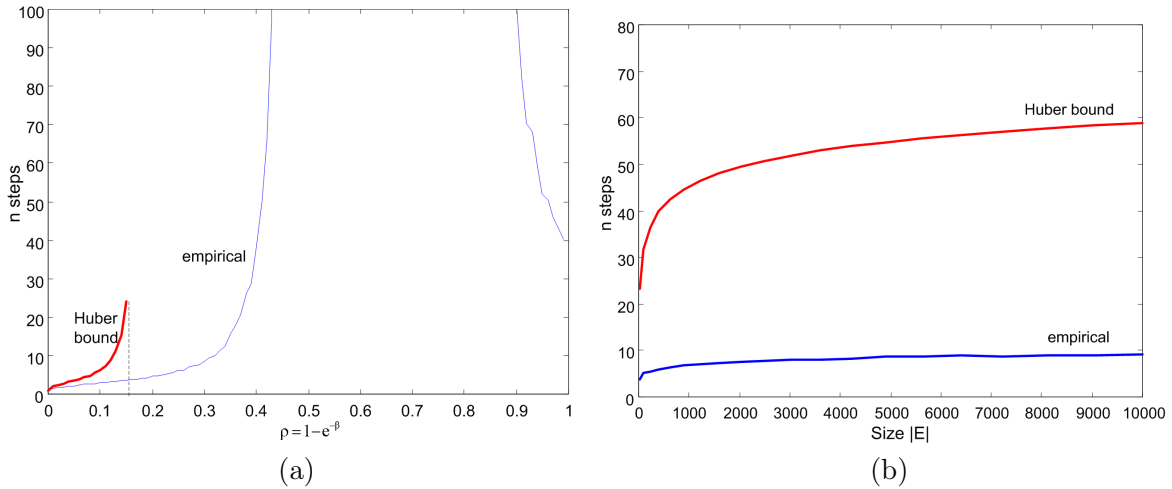


Figure 6.5: The coupling time empirical plots and the Huber bounds for Ising model.

Despite the encouraging success discussed above, the SW method is limited in two aspects.

Limit 1. It is only valid for the Ising and Potts models, and furthermore it requires that the number of colorings L is known. In many applications, such as image analysis, L is the number of objects (or image regions) which has to be inferred from the input data.

Limit 2. It slows down quickly in the presence of external field, i.e input data. For example, in the image analysis problem, our goal is to infer the label \mathbf{X} from the input image \mathbf{I} and the target

probability is a Bayesian posterior probability where $\pi_{\text{PTS}}(\mathbf{X})$ is used as a prior model,

$$\pi(\mathbf{X}) = \pi(\mathbf{X}|\mathbf{I}) \propto \mathcal{L}(\mathbf{I}|\mathbf{X})\pi_{\text{PTS}}(\mathbf{X}). \quad (6.36)$$

$\mathcal{L}(\mathbf{I}|\mathbf{X})$ is the likelihood model, such as independent Gaussians $N(\bar{\mathbf{I}}_c, \sigma_c^2)$ for each coloring $c = 1, 2, \dots, L$,

$$\mathcal{L}(\mathbf{I}|\mathbf{X}) \propto \prod_{c=1}^L \prod_{x_i=c} \frac{1}{\sqrt{2\pi}\sigma_c} \exp\left\{-\frac{(\mathbf{I}(v_i) - \bar{\mathbf{I}}_c)^2}{2\sigma_c^2}\right\}. \quad (6.37)$$

The slowing down is partially attributed to the fact that the Bernoulli probability $\rho = 1 - e^{-\beta}$ for the auxiliary variable is calculated independently of the input image.

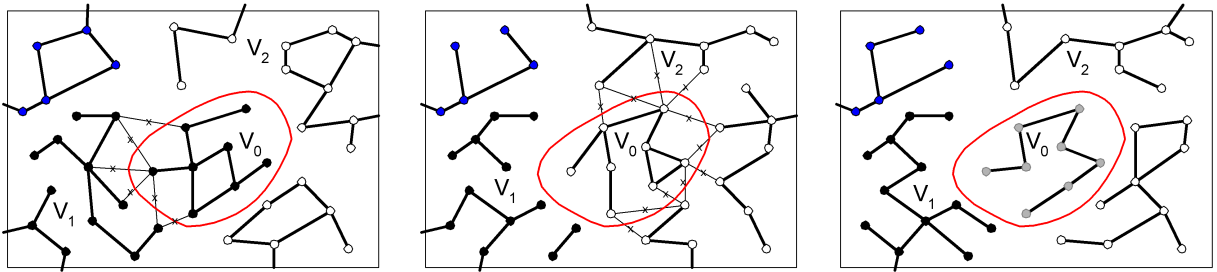


Figure 6.6: A reversible move between three partition states π_A (left), π_B (middle), and π_C (right) that differ only in the color of the set V_0 . The vertices connected by thick edges form a connected component. The thin lines marked with crosses are edges in the SW-cuts.

6.4 Swendsen-Wang Cuts for Arbitrary Probabilities

In this section, we generalize the SW to arbitrary probabilities from the perspective of Metropolis-Hastings method [91, 139]. Our method iterates three steps: (i) a clustering step driven by data, (ii) a label flipping step which can introduce new labels, and (iii) an acceptance step for the proposed labelling. A key observation is the simple formula in calculating the acceptance probabilities.

We deliberate the three steps in the following three subsections, and then we show how it reduces to the original SW with Potts models.

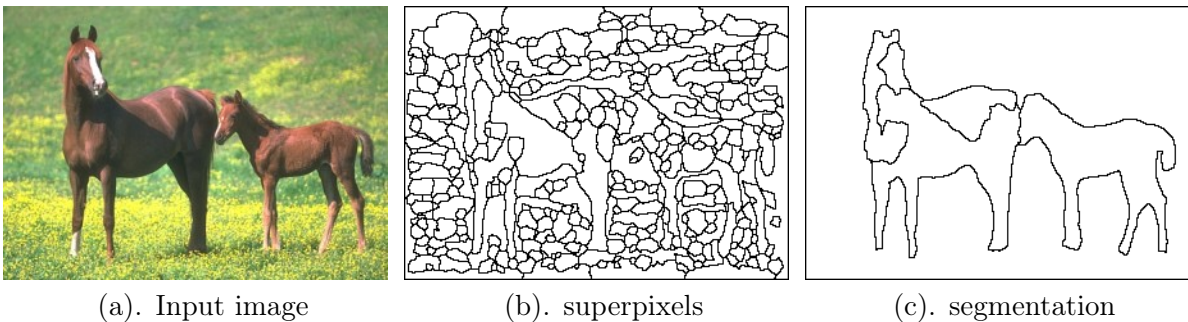


Figure 6.7: Example of image segmentation. (a). Input image. (b). Superpixels obtained by edge detection followed by edge tracing and contour closing. Each superpixel is a vertex in the graph \mathbf{G} . c. Segmentation (labeling) result where each closed region is assigned a color or label.

We illustrate the algorithm by an example on image segmentation shown in Fig. 6.7. Fig. 6.7.(a) is an input image \mathbf{I} on a lattice Λ , which is decomposed into a number of "superpixels" to reduce the graph size in a preprocessing stage. Each superpixel has nearly constant intensity and is a vertex in the graph \mathbf{G} . Two vertices are connected if their superpixels are adjacent (i.e. sharing boundary). Fig. 6.7.(c) is a result by our algorithm optimizing a Bayesian probability $\pi(\mathbf{X}) = \pi(\mathbf{X}|\mathbf{I})$ (see section (6.6) for details). The result \mathbf{X} assigns a uniform color to all vertices in each close region which hopefully corresponds to an object in the scene or a part of it. Note that the number of objects or colors L is unknown, and we do not distinguish the different permutations of the labels.

6.4.1 Step 1: data-driven clustering

We augment the adjacency graph \mathbf{G} with a set of binary variables on the edges $\mathbf{U} = \{\mu_e : e = \langle s, t \rangle \in E\}$, as in the original SW method. Each μ_e follows a Bernoulli probability depending on the current state of the two vertices x_s and x_t ,

$$\mu_e|(x_s, x_t) \sim \text{Bernoulli}(q_e \mathbf{1}(x_s = x_t)), \quad \forall \langle s, t \rangle \in E. \quad (6.38)$$

q_e is a probability on edge $e = \langle s, t \rangle$ which tells how likely the two vertices s and t have the same label. In Bayesian inference where the target $\pi(\mathbf{X})$ is a posterior probability, then q_e can be better informed by the data.

For the image segmentation example, q_e is computed based on the similarity between image intensities at s and t (or their local neighborhood) and it may be an approximate to the marginal probability of $\pi(\mathbf{X}|\mathbf{I})$,

$$q_e = q(x_s = x_t | \mathbf{I}(s), \mathbf{I}(t)) \approx \pi(x_s = x_t | \mathbf{I}). \quad (6.39)$$

There are many ways for computing $q(x_s = x_t | \mathbf{I}(v_s), \mathbf{I}(v_t))$ using so called discriminative methods, and it is beyond the scope of this book to discuss details.



Figure 6.8: Three examples of the connected components for the horse image computed using discriminative edge probabilities given that \mathbf{X} is a uniform color $\mathbf{X} = c$ for all vertices.

Our method will work for any q_e , but a good approximation will inform the clustering step and achieve faster convergence empirically. Fig. 6.8 shows nine clustering examples of the horse image. In these examples, we set all vertices to the same color ($\mathbf{X} = c$) and sample the edge probability independently,

$$\mathbf{U} | \mathbf{X} = c \sim \prod_{\langle s, t \rangle \in E} \text{Bernoulli}(q_e). \quad (6.40)$$

The connected components in $\text{CP}(\mathbf{U})$ are shown by different regions. We repeat the clustering step nine times. As we can see, the edge probabilities lead to "meaningful" clusters which correspond to distinct objects in the image. Such effects cannot be observed using constant edge probabilities.

6.4.2 Step 2: flipping of color

Let $\mathbf{X} = (x_1, x_2, \dots, x_{|V|})$ be the current coloring state, and the edge variables \mathbf{U} sampled conditional on \mathbf{X} further decompose \mathbf{X} into a number of connected components

$$\text{CP}(\mathbf{U}|\mathbf{X}) = \{\text{cp}_i : i = 1, 2, \dots, N(\mathbf{U}|\mathbf{X})\}. \quad (6.41)$$

Suppose we select one connected component $V_o \in \text{CP}(\mathbf{U}|\mathbf{X})$ with color $\mathbf{X}_{V_o} = \ell \in \{1, 2, \dots, L\}$, and assign its color to $\ell' \in \{1, 2, \dots, L, L+1\}$ with probability $q(\ell'|V_o, \mathbf{X})$ (to be designed shortly), obtaining new state \mathbf{X}' . There are three cases shown in Fig. 6.3.

1. The canonical case: $V_o \subset V_\ell$ and $\ell' \leq L$. That is, a portion of V_ℓ is re-grouped into an existing color $V_{\ell'}$, and the number of colors remains $L = L$ in π' . The moves between $A \leftrightarrow B$ in Fig. 6.3 are examples.
2. The merge case: $V_o = V_\ell$ in \mathbf{X} is the set of all vertices that have color ℓ and $\ell' \leq L$, $\ell \neq \ell'$. That is, color V_ℓ is merged to $V_{\ell'}$, and the number of distinct colors reduces to $L - 1$ in \mathbf{X}' . The moves $\mathbf{X}_C \rightarrow \mathbf{X}_A$ or $\mathbf{X}_C \rightarrow \mathbf{X}_B$ in Fig. 6.3 are examples.
3. The split case: $V_o \subset V_\ell$ and $\ell' = L + 1$. V_ℓ is split into two pieces and the number of distinct color increases to $L + 1$ in \mathbf{X}' . The moves $\mathbf{X}_A \rightarrow \mathbf{X}_C$ in Fig. 6.3 are examples.

Note that this color flipping step is also different from the original SW with Potts model as we allow new colors in each step. The number of color L is not fixed.

6.4.3 Step 3: accepting the flipping

The previous two steps basically have proposed a move between two states \mathbf{X} and \mathbf{X}' which differ in coloring a connected component V_o . In the third step we accept the move with probability,

$$\alpha(\mathbf{X} \rightarrow \mathbf{X}') = \min\{1, \frac{q(\mathbf{X}' \rightarrow \mathbf{X})}{q(\mathbf{X} \rightarrow \mathbf{X}')} \cdot \frac{\pi(\mathbf{X}')}{\pi(\mathbf{X})}\}. \quad (6.42)$$

$q(\mathbf{X}' \rightarrow \mathbf{X})$ and $q(\mathbf{X} \rightarrow \mathbf{X}')$ are the proposal probabilities between \mathbf{X} and \mathbf{X}' . If the proposal is rejected, the Markov chain stays at state \mathbf{X} . The transition kernel is

$$\mathcal{K}(\mathbf{X} \rightarrow \mathbf{X}') = q(\mathbf{X} \rightarrow \mathbf{X}')\alpha(\mathbf{X} \rightarrow \mathbf{X}'), \quad \forall \mathbf{X} \neq \mathbf{X}'. \quad (6.43)$$

For the canonical case, there is a unique path for moving between \mathbf{X} and \mathbf{X}' in one step – choosing V_o and changing its color. The proposal probability ratio is the product of two ratios decided by the clustering and flipping steps respectively: (i) the probability ratio for selecting V_o as the candidate in the clustering step in both states \mathbf{X} and \mathbf{X}' , and (ii) the probability ratio for selecting the new labels for V_o in the flipping step.

$$\frac{q(\mathbf{X}' \rightarrow \mathbf{X})}{q(\mathbf{X} \rightarrow \mathbf{X}')} = \frac{q(V_o|\mathbf{X}')}{q(V_o|\mathbf{X})} \cdot \frac{q(\mathbf{X}_{V_o} = \ell|V_o, \mathbf{X}')}{q(\mathbf{X}_{V_o} = \ell'|V_o, \mathbf{X})}. \quad (6.44)$$

For the split and merge cases, there are two paths between \mathbf{X} and \mathbf{X}' . But this does not change the conclusion (see Appendix B).

Now we compute the probability ratio $\frac{q(V_o|\mathbf{X}')}{q(V_o|\mathbf{X})}$ for proposing V_o .

Definition 6.1. Let $\mathbf{X} = (V_1, V_2, \dots, V_L)$ be a coloring state, and $V_o \in \text{CP}(U|\mathbf{X})$ a connected component, the "cut" between V_o and V_k is a set of edges between V_o and $V_k \setminus V_o$,

$$\mathcal{C}(V_o, V_k) = \{ \langle s, t \rangle : s \in V_o, t \in V_k \setminus V_o \}, \quad \forall k.$$

One of the key observations is that this ratio only depends on the cuts between V_o and rest vertices.

Proposition 6.7. *In the above notation, we have*

$$\frac{q(V_o|\mathbf{X})}{q(V_o|\mathbf{X}')} = \frac{\prod_{\langle i,j \rangle \in \mathcal{C}(V_o, V_\ell)} (1 - q_{ij})}{\prod_{\langle i,j \rangle \in \mathcal{C}(V_o, V_{\ell'})} (1 - q_{ij})}. \quad (6.45)$$

q_e 's are the edge probabilities.

Thus the acceptance probability is given by the following

Theorem 6.8. (Barbu and Zhu, 2005) *The acceptance probability for the proposed swapping is*

$$\alpha(\mathbf{X} \rightarrow \mathbf{X}') = \min\left\{1, \frac{\prod_{\langle i,j \rangle \in \mathcal{C}(V_o, V_{\ell'})} (1 - q_{ij})}{\prod_{\langle i,j \rangle \in \mathcal{C}(V_o, V_\ell)} (1 - q_{ij})} \cdot \frac{q(\mathbf{X}_{V_o} = \ell | V_o, \mathbf{X}')}{q(\mathbf{X}_{V_o} = \ell' | V_o, \mathbf{X})} \cdot \frac{\pi(\mathbf{X}')}{\pi(\mathbf{X})}\right\}. \quad (6.46)$$

[Proof] See Theorem 2 in Barbu and Zhu [9].

Example 6.1. In image analysis, $\pi(\mathbf{X})$ is a Bayesian posterior $\pi(\mathbf{X}|\mathbf{I}) \propto \mathcal{L}(\mathbf{I}|\mathbf{X})p_o(\mathbf{X})$ with the prior probability $p_o(\mathbf{X})$ being a Markov model (like Potts in Eqn. (6.37)). Therefore one can compute the ratio of the target probabilities in the local neighborhood of V_o , i.e. ∂V_o .

$$\frac{\pi(\mathbf{X}')}{\pi(\mathbf{X})} = \frac{\mathcal{L}(\mathbf{I}_{V_o}|\mathbf{X}_{V_o} = \ell')}{\mathcal{L}(\mathbf{I}_{V_o}|\mathbf{X}_{V_o} = \ell)} \cdot \frac{p_o(\mathbf{X}_{V_o} = \ell' | \mathbf{X}_{\partial V_o})}{p_o(\mathbf{X}_{V_o} = \ell | \mathbf{X}_{\partial V_o})} \quad (6.47)$$

Note that $\mathbf{X}_{\partial V_o} = \mathbf{X}'_{\partial V_o}$ in the above equation.

The second ratio in eq.(6.46) is easy to design. For example, we can make it proportional to the likelihood,

$$q(\mathbf{X}_{V_o} = \ell | V_o, \mathbf{X}) = \mathcal{L}(\mathbf{I}_{V_o}|\mathbf{X}_{V_o} = \ell), \quad \forall \ell. \quad (6.48)$$

Therefore,

$$\frac{q(\mathbf{X}_{V_o} = \ell | V_o, \mathbf{X}')}{q(\mathbf{X}_{V_o} = \ell' | V_o, \mathbf{X})} = \frac{\mathcal{L}(\mathbf{I}_{V_o}|\mathbf{X}_{V_o} = \ell)}{\mathcal{L}(\mathbf{I}_{V_o}|\mathbf{X}_{V_o} = \ell')} \quad (6.49)$$

It cancels the likelihood ratio in eqn.(6.47). We get

Proposition 6.9. *The acceptance probability for the proposed cluster flipping using the proposal (6.48) is,*

$$\alpha(\mathbf{X} \rightarrow \mathbf{X}') = \min\left\{1, \frac{\prod_{\langle s,t \rangle \in \mathcal{C}(R, V_{\ell'})} (1 - q_e)}{\prod_{e \in \mathcal{C}(V_o, V_\ell)} (1 - q_e)} \cdot \frac{p_o(\mathbf{X}_{V_o} = \ell' | \mathbf{X}_{\partial V_o})}{p_o(\mathbf{X}_{V_o} = \ell | \mathbf{X}_{\partial V_o})}\right\}. \quad (6.50)$$

The result above states that the computation is limited to a local neighborhood of V_o defined by the prior model. This result is also true if one changes the clustering step by growing V_o from a vertex, i.e. the Wolff modification.

In the experiments on image analysis, our cluster sampling method is empirically $O(100)$ times faster than the single site Gibbs sampler in terms of CPU time. We refer to plots and comparison in Figs.(6.11), (6.12) and (6.13) in section (6.6) for details.

6.4.4 Complexity Analysis

This section presents an evaluation of the computation complexity of the SWC algorithm.

Let $N = |V|$ be the number of points of the graph $\mathbf{G} = \langle V, E \rangle$.

Each of the N^{it} iterations of the SWC algorithm involves the following steps:

- Sampling the edges in the data driven clustering step, which is $O(|E|)$. Assuming that $\mathbf{G} = \langle V, E \rangle$ is sparse (which is usually the case in vision), then $O(|E|) = O(N)$.
- Constructing connected components, which is $O(|E|\alpha(|E|)) = O(N\alpha(N))$ using the disjoint set forest data structure [66, 70]. The function $\alpha(N)$ is the inverse of $f(n) = A(n, n)$ where $A(m, n)$ is the fast growing Ackerman function [1]. In fact $\alpha(N) \leq 5$ for $N \leq 2^{2^{10^{19729}}}$, thus for all practical values of N .
- Computing $\pi(\mathbf{X})$ depends on the problem, but is usually $O(N)$.
- Flipping the label of one connected component, which is $O(N)$.

Therefore one iteration is $O(N\alpha(N))$ and all the SWC iterations take $O(N^{it}N\alpha(N))$ time.

6.4.5 Example: SWC for Gaussian Mixture Models

Let $\{\mathbf{x}_i \in \mathbb{R}^d, i = 1, \dots, n\}$ be data points assumed to originate from a mixture of k multivariate Gaussians with unknown mixture weights α_i , means $\mu_i \in \mathbb{R}^d$ and covariance matrices Σ_i , for $i = 1, \dots, K$. Let Θ contain all the unknown mixture parameters $\alpha_i, \mu_i, \Sigma_i, i = 1, \dots, K$.

The log likelihood (energy) of the Gaussian mixture model is:

$$\log P(\Theta) = \sum_{i=1}^n \log \sum_{j=1}^K \alpha_j G(\mathbf{x}_i; \mu_j, \Sigma_j) - \log Z(\Theta), \quad (6.51)$$

where $G(\mathbf{x}_i; \mu_j, \Sigma_j) = \frac{1}{\sqrt{\det(2\pi\Sigma_j)}} \exp \left[-\frac{1}{2} (\mathbf{x}_i - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \mu_j) \right]$ is a Gaussian and $Z(\theta)$ is the normalization constant.

If the labels assigning points to clusters are known, say $L = (l_1, \dots, l_n)$ then the log likelihood is

$$\log P(L, \Theta) = \sum_{j=1}^K \sum_{i \in L_j} \log G(\mathbf{x}_i; \mu_j, \Sigma_j)$$

where $L_j = \{i, l_i = j\}$.

Sampling $P(\Theta)$ can be done by sampling $P(L, \Theta)$ and taking the marginal $P(\theta)$. Sampling $P(L, \Theta)$ can be done by alternating sampling $P(L|\Theta)$ and $P(\Theta|L)$.

For sampling $P(L|\Theta)$ we can use the SWC algorithm. We construct the SWC graph as the k -NN graph and use a constant probability q for all the edge weights.

Sampling $P(\Theta|L)$ is more involved. First, we should observe that $P(\Theta|L) = \prod_{j=1}^K \prod_{i \in L_j} G(\mathbf{x}_i; \mu_j, \Sigma_j)$

splits in independent parts: $P(\Theta|L) = \prod_{j=1}^K P(\Theta_j|L_j)$, where $\theta_j = (\alpha_j, \mu_j, \Sigma_j)$. Thus we can sample $P(\Theta_j|L_j)$ independently for each j by sampling $P(\mu_j|L_j, \Sigma_j)$ and $P(\Sigma_j|\mu_j, L_j)$. Now

$$P(\mu_j|\Sigma_j, L_j) = \prod_{i \in L_j} G(\mathbf{x}_i; \mu_j, \Sigma_j) \propto G(\mu_j, \frac{1}{n_j} \sum_{i \in L_j} \mathbf{x}_i, \frac{1}{n_j} \Sigma_j)$$

is just a Gaussian, where $n_j = |L_j|$. Also,

$$P(\Sigma_j | \mu_j, L_j) = \det(\Sigma_j)^{-n_j/2} \exp\left(-\frac{1}{2} \sum_{i \in L_j} (\mu_j - \mathbf{x}_i)^T \Sigma_j^{-1} (\mu_j - \mathbf{x}_i)\right) = \det(\Sigma_j)^{-n_j/2} \exp\left(-\frac{1}{2} \text{tr}(\hat{\Sigma} \Sigma_j^{-1})\right)$$

where $\hat{\Sigma} = \sum_{i \in L_j} (\mu_j - \mathbf{x}_i)(\mu_j - \mathbf{x}_i)^T$ and we used that $\text{tr}(AB) = \text{tr}(BA)$ with $A = (\mu_j - \mathbf{x}_i)$ and $B = (\mu_j - \mathbf{x}_i)^T \Sigma_j^{-1}$. Since $\hat{\Sigma}$ is symmetric and positive definite, there exists symmetric positive definite S such that $\hat{\Sigma} = S^2$. Then writing $B = S \Sigma_j^{-1} S$ we get

$$P(\Sigma_j | \mu_j, L_j) = \det(\Sigma_j)^{-n_j/2} \exp\left(-\frac{1}{2} \text{tr}(S \Sigma_j^{-1} S)\right) = \det(S)^{-n_j/2} \det(B)^{n_j/2} \exp\left(-\frac{1}{2} \text{tr}(B)\right).$$

Now writing $B = UDU^T$ where $D = \text{diag}(\lambda_1, \dots, \lambda_d)$ is diagonal we obtain

$$P(\Sigma_j | \mu_j, L_j) \propto \det(D)^{n_j/2} \exp\left(-\frac{1}{2} \text{tr}(D)\right) = \prod_{i=1}^d \lambda_i^{n_j/2} e^{-\lambda_i/2}$$

so to sample Σ_j we first sample the eigenvalues λ_i independently from the Gamma distribution $\Gamma(1 + \frac{n_j}{2}, 2)$ to obtain $D = \text{diag}(\lambda_1, \dots, \lambda_d)$, then take any rotation matrix U to obtain $B = UDU^T$ and $\Sigma_j = SUDU^T S$.

6.5 Variants of the cluster sampling method

In this section, we briefly discuss two variants of the cluster sampling method.

6.5.1 Cluster Gibbs sampling — the "hit-and-run" perspective

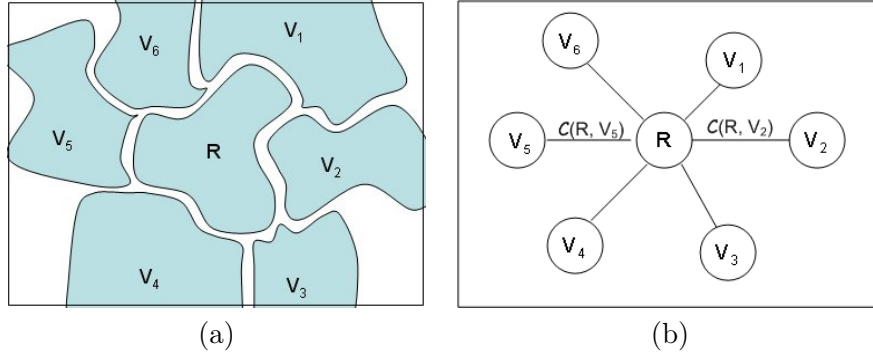


Figure 6.9: Illustrating the cluster Gibbs sampler. (a) The cluster V_o has a number of neighboring components of uniform color. (b) The cuts between V_o and its neighboring colors. The sampler follows a conditional probability modified by the edge strength defined on the cuts.

With a slight change, we can modify the cluster sampling method to become a generalized Gibbs sampler.

Suppose that $V_o \in \text{CP}(U|\mathbf{X})$ is the candidate chosen in the clustering step, and Fig. 6.9 shows its cuts with adjacent sets

$$\mathcal{C}(V_o, V_k), k = 1, 2, \dots, L(\mathbf{X}).$$

We compute the edge weight γ_k as the strength of connectivity between V_o and $V_k \setminus R$,

$$\gamma_k = \prod_{e \in \mathcal{C}(V_o, V_k)} (1 - q_e). \quad (6.52)$$

Proposition 6.10. *Let $\pi(\mathbf{X})$ be the target probability, in the notation above. If R is relabeled probabilistically with*

$$q(\mathbf{X}_{V_o} = k | V_o, \mathbf{X}) \propto \gamma_k \pi(\mathbf{X}_{V_o} = k | \mathbf{X}_{\partial V_o}), \quad k = 1, 2, \dots, N(\mathbf{X}), \quad (6.53)$$

then the acceptance probability is always 1 in the third step.

This yields a generalized Gibbs sampler which flips the color of a cluster according to a modified conditional probability.

Cluster Gibbs Sampler

1. Cluster step: choosing a vertex $v \in V$ and group a cluster V_o from v by the Bernoulli edge probability μ_e .
2. Flipping step: relabel V_o according to eqn. (6.53).

The traditional single site Gibbs sampler [75] is a special case when $q_e = 0$ for all e and thus $V_o = \{v\}$ and $\gamma_k = 1$ for all k .

One may also view the above method from the perspective of hit-and-run. In continuous state space, a hit-and-run method (see [84]) chooses a new direction \vec{e} (random ray) at time t and then sample on this direction by $a \sim \pi(x + a\vec{e})$. Liu and Wu [125] extended it ray to any compact groups of actions. In finite state space Ω , one can choose any finite sets $\Omega_a \subset \Omega$ and then apply the Gibbs sampler within the set.

But it is difficult to choose good directions or subsets in hit-and-run methods. In the cluster Gibbs sampler presented above, the subset is selected by the auxiliary variables on the edges.

6.5.2 The multiple flipping scheme

Given a set of connected components $\text{CP}(\mathbf{U}|\mathbf{X})$ (see eqn. (6.41)) after the clustering step, instead of flipping a single component R , we can flip all (or any chosen number of) connected components simultaneously. There is room for designing the proposal probabilities for labeling these connected components, independently or jointly. In what follows, we assume the labels are chosen independently for each connected component $\text{cp} \in \text{CP}(\mathbf{U}|\mathbf{X})$, by sampling from a proposal probability $q(\mathbf{X}_{\text{cp}} = l | \text{cp})$. Suppose we obtain a new label \mathbf{X}' after flipping. Let $E_{\text{on}}(\mathbf{X}) \subset E$ and $E_{\text{on}}(\mathbf{X}') \subset E$ be the subsets of edges that connect the vertices of same color in \mathbf{X} and \mathbf{X}' respectively. We define two cuts by the differences of the sets

$$\mathcal{C}(\mathbf{X} \rightarrow \mathbf{X}') = E_{\text{on}}(\mathbf{X}') - E_{\text{on}}(\mathbf{X}), \quad \text{and} \quad \mathcal{C}(\mathbf{X}' \rightarrow \mathbf{X}) = E_{\text{on}}(\mathbf{X}) - E_{\text{on}}(\mathbf{X}'), \quad (6.54)$$

We denote the set of connected components which have different colors before and after the flipping by $D(\mathbf{X}, \mathbf{X}') = \{\text{cp} : \mathbf{X}_{\text{cp}} \neq \mathbf{X}'_{\text{cp}}\}$.

Proposition 6.11. *The acceptance probability of the multiple flipping scheme is*

$$\alpha(\mathbf{X} \rightarrow \mathbf{X}') = \min\left\{1, \frac{\prod_{e \in \mathcal{C}(\mathbf{X} \rightarrow \mathbf{X}')} (1 - q_e) \prod_{\text{cp} \in D(\mathbf{X}, \mathbf{X}')} q(\mathbf{X}'_{\text{cp}} | \text{cp})}{\prod_{e \in \mathcal{C}(\mathbf{X}' \rightarrow \mathbf{X})} (1 - q_e) \prod_{\text{cp} \in D(\mathbf{X}, \mathbf{X}')} q(\mathbf{X}_{\text{cp}} | \text{cp})} \cdot \frac{p(\pi')}{p(\pi)}\right\} \quad (6.55)$$

Observe that when $D = \{V_o\}$ is a single connected component, this reduces to Theorem 6.50.

It is worth mentioning that if we flip all connected components simultaneously, then the Markov transition graph of $\mathcal{K}(\mathbf{X}, \mathbf{X}')$ is fully connected, i.e.

$$\mathcal{K}(\mathbf{X}, \mathbf{X}') > 0, \quad \forall \mathbf{X}, \mathbf{X}' \in \Omega. \quad (6.56)$$

This means that the Markov chain can walk between any two partitions in a single step.

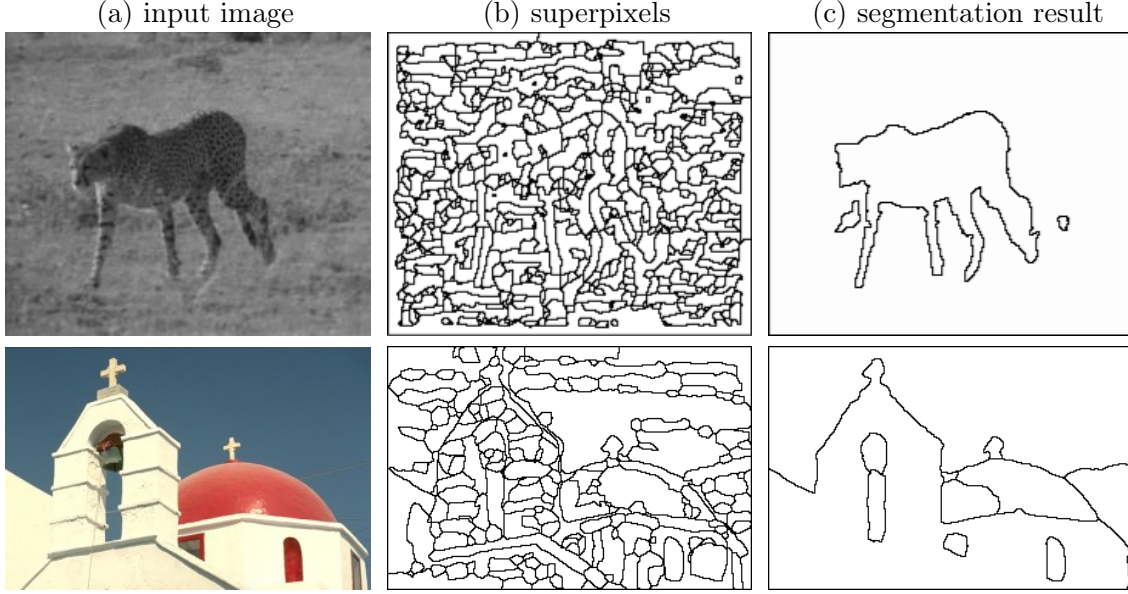


Figure 6.10: More results for image segmentation.

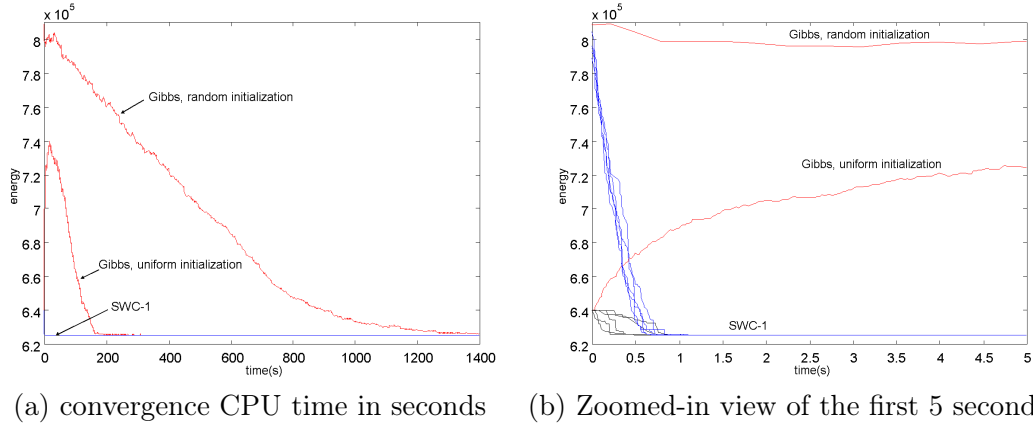


Figure 6.11: The plot of $-\ln \pi(X)$ over computing time for both the Gibbs sampler and our algorithm for the horse image. Both algorithms are measured by the CPU time in seconds using a Pentium IV PC. So they are comparable. (a). Plot in the first 1,400 seconds. The Gibbs sampler needs a high initial temperature and slow annealing step to achieve the same energy level. (b). The zoomed-in view of the first 5 seconds.

6.6 Application: Image Segmentation

Our first experiment tests the cluster sampling algorithm in an image segmentation task. The objective is to partition the image into a number of disjoint regions (as Figs.6.7 and 6.8 have shown) so that each region has consistent intensity in the sense of fitting to some image models. The final result should optimize a Bayesian posterior probability $\pi(\mathbf{X}) \propto \mathcal{L}(\mathbf{I}|\mathbf{X})p_o(\mathbf{X})$.

In such problem, \mathbf{G} is an adjacency graph with vertices V being a set of superpixels (see Figs.(6.7) and (6.8)). Usually $|V| = O(10^2)$. For each superpixel $v \in V$, we compute a 15-bin intensity histogram h normalized to 1. Thus the edge probability is calculated as

$$q_{ij} = p(\mu_e = \text{on}|\mathbf{I}(v_i), \mathbf{I}(v_j)) = \exp\left\{-\frac{1}{2}(KL(h_i||h_j) + KL(h_j||h_i))\right\}, \quad (6.57)$$

where $KL()$ is the Kullback-Leibler divergence between the two histograms. Usually q_e should be close to zero for e crossing object boundary. In our experiments, the edge probability leads to good clustering as Fig. 6.8 shows.

Now we briefly define the target probability in this experiment. Let $\mathbf{X} = (V_1, \dots, V_L)$ be a coloring of the graph with L being a unknown variable, and the image intensities in each set V_k is consistent in terms of fitting to a model θ_k . Different colors are assumed to be independent. Therefore, we have,

$$\pi(\mathbf{X}) = \pi(\mathbf{X}|\mathbf{I}) \propto \prod_{k=1}^L [\mathcal{L}(\mathbf{I}(V_k); \theta_k) p_o(\theta_k)] p_o(\mathbf{X}). \quad (6.58)$$

We selected three types of simple models for the likelihood models to account for different image properties. The first model is a non-parametric histogram \mathcal{H} , which in practice is represented by a vector of B -bins $(\mathcal{H}_1, \dots, \mathcal{H}_B)$ normalized to 1. It accounts for cluttered objects, like vegetation.

$$\mathbf{I}(x, y; \theta_0) \sim \mathcal{H} \text{ iid}, \forall (x, y) \in V_k. \quad (6.59)$$

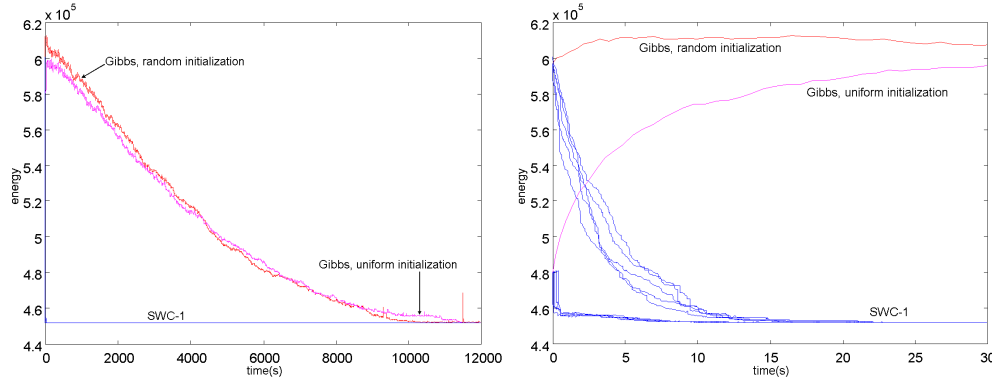


Figure 6.12: Convergence comparison between the clustering method and Gibbs sampler in CPU time (seconds) on the artificial image (circles, triangle and rectangles) in the first row of Fig.6.10. (Left). The first 1,200 seconds. (Right) Zoomed-in view of the first 30 seconds. The clustering algorithm is run 5 trials for both the random and uniform initializations.

The other two are regression models for the smooth change of intensities in the two-dimensional image plane (x, y) , and the residues follow the empirical distribution \mathcal{H} (i.e. the histogram).

$$\mathbf{I}(x, y; \theta_1) = \beta_0 + \beta_1 x + \beta_2 y + \mathcal{H} \text{ iid}, \forall (x, y) \in V_k. \quad (6.60)$$

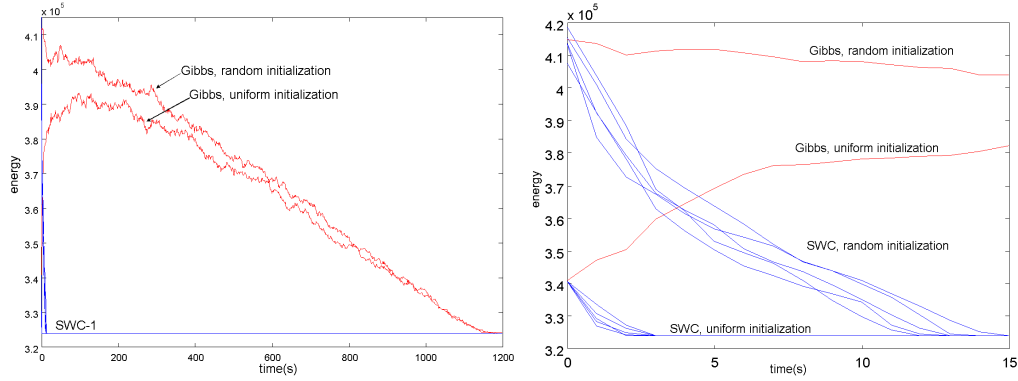


Figure 6.13: Convergence comparison between the clustering method and Gibbs sampler in CPU time (seconds) on the cheetah image. (Left) The first 1,200 seconds. (Right) Zoomed-in view of the first 15 seconds. The clustering algorithm is run 5 times for both the random and uniform initializations.

$$\mathbf{I}(x, y; \theta_2) = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 x^2 + \beta_4 xy + \beta_5 y^2 + \mathcal{H} \text{ iid}, \forall (x, y) \in V_k. \quad (6.61)$$

In all cases, the likelihood is expressed in terms of the entropy of the histogram \mathcal{H}

$$\mathcal{L}(\mathbf{I}(V_k); \theta_k) \propto \prod_{v \in V_k} \mathcal{H}(\mathbf{I}_v) = \prod_{j=1}^B \mathcal{H}_j^{n_j} = \exp(-|V_k| \text{entropy}(\mathcal{H})). \quad (6.62)$$

The model complexity is penalized by a prior probability $p_o(\theta_k)$ and the parameters θ in the above likelihoods are computed deterministically at each step as the best least square fit. The deterministic fitting could be replaced by the reversible jumps together with the flipping of color. This was done in [193] and will be presented in Chapter 8.

The prior model $p_o(\mathbf{X})$ encourages large and compact regions with small number of colors, as it was suggested in [193]. Let r_1, r_2, \dots, r_m , $m \geq L$ be the connected components of all $V_k, k = 1, \dots, L$. Then the prior is

$$p_o(\mathbf{X}) \propto \exp\{-\alpha_0 L - \alpha_1 m - \alpha_2 \sum_{k=1}^m \text{Area}(r_k)^{0.9}\}. \quad (6.63)$$

For the image segmentation example (horse) shown in Figs. 6.7 and 6.8, we compare the cluster sampling method with the single-site Gibbs sampler and the results are displayed in Fig. 6.11. Since our goal is to maximize the posterior probability $\pi(\mathbf{X})$, we must add an annealing scheme with a high initial temperature T_o and then decreases to a low temperature (0.05 in our experiments). We plot the $-\ln \pi(\mathbf{X})$ over CPU time in seconds with a Pentium IV PC. The Gibbs sampler needs to raise the initial temperature high (say $T_o \geq 100$) and uses a slow annealing schedule to reach a good solution. The cluster sampling method can run at low temperature. We usually raise the initial temperature to $T_o \leq 15$ and use a fast annealing scheme. Fig. 6.11.(a) plots the two algorithms at the first 1,400 seconds, and Fig. 6.11.(b) is a zoomed-in view for the first 5 seconds.

We run the two algorithms with two initializations. One is a random labeling of the superpixels and thus has higher $-\ln \pi(\mathbf{X})$, and the other initialization sets all vertices to the same color. The clustering methods are run five times on both cases. They all converged to one solution (see Fig.6.7.(c)) within 1 second, which is $O(10^2)$ times faster than the Gibbs sampler.

Fig.6.10 shows four more images. Using the sample comparison method as in the horse image, we plot $-\ln \pi(\mathbf{X})$ against running time in Figs. 6.12 and 6.13 for the images in the first and second row of Fig.6.10 respectively. In experiments, we also compared the effect of the edge probabilities. The clustering algorithm are $O(100)$ times slower if we use a constant edge probability $\mu_{ij} = c \in (0, 1)$ as the original SW method does. For example the single-site Gibbs sampler is an example with $q_{ij} = 0, \forall i, j$.

6.7 Multigrid and multi-level SW-cut

The essence of the SW-cut is a Markov chain $\mathcal{MC} = \langle \nu, \mathcal{K}, p \rangle$ which visits a sequence of states in the partition space Ω_π over time t ,

$$\pi(0), \pi(1), \dots, \pi(t) \in \Omega_\pi.$$

The \mathcal{MC} consists of three elements. (1). An initial probability $\nu(\pi)$ with $\pi(0) \sim \nu(\pi)$. (2). A transition kernel $\mathcal{K}(\pi, \pi')$ which is a conditional probability for moving from state $\pi(t) = \pi$ to $\pi(t+1) = \pi'$. (3). An invariance probability $p(\pi)$ which is the Bayesian posterior probability for the partitions in a vision task.

The three theorems in the previous section ensure that the SW-cut design observes the detailed balance equations

$$p(\pi)\mathcal{K}(\pi, \pi') = p(\pi')\mathcal{K}(\pi', \pi), \quad \forall \pi', \pi. \quad (6.64)$$

This is a sufficient condition for $p(\pi)$ being the invariant probability of the kernel \mathcal{K} ,

$$\sum_{\pi} p(\pi)\mathcal{K}(\pi, \pi') = p(\pi'). \quad (6.65)$$

Once it converges, the SW-cut simulates fair samples from $p(\pi)$

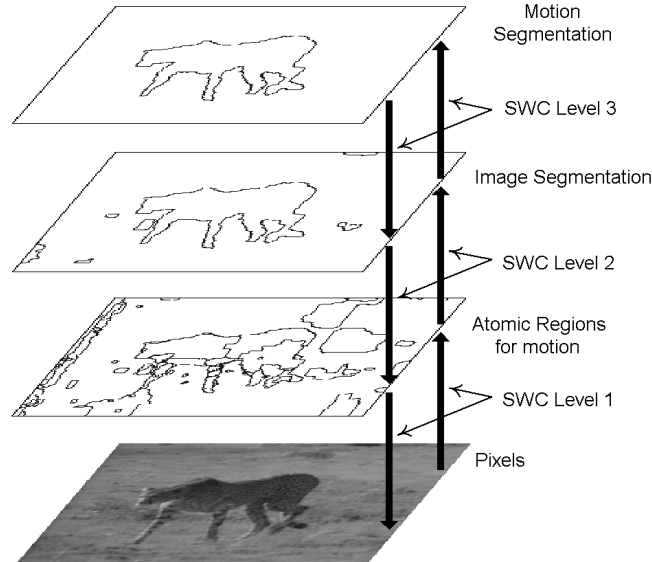


Figure 6.14: Multi-level SW-cut for joint image and motion segmentation.

The SW-cut is characterized by three selections in its design.

(I). The discriminative proposal probabilities defined on the edges of the adjacency graph $G = \langle V, E \rangle$. $q(\pi) = \prod_{e \in E} q_e$ is a factorized approximation to $p(\pi)$ and it influences the formation connected components CP, and thus the candidate component V_o .

(II) The uniform probability for selecting V_o from a connected component $V_o \in \text{CP}$.

(III) The reassignment probability $Q(\ell_{\text{new}}(V_o)|V_o, \pi_A)$ for the new label of the connected component V_o .

We extend the SW-cut for speed and generality by introducing the multigrid and multilevel SW-cuts which provide more flexible means for selecting V_o 's and $q(\pi)$'s.

In summary, the two extensions are new directions for sampling $p(\pi)$

1. The multigrid SW-cuts simulates Markov chain \mathcal{MC}_{mg} with kernel \mathcal{K}_{mg} sampling the conditional probabilities of $p(\pi)$.
2. The multi-level SW-cuts simulates Markov chain \mathcal{MC}_{ml} with kernel \mathcal{K}_{ml} sampling the conditional probabilities of $p(\pi)$ at the higher levels, and the full posterior at the lower level.

Both \mathcal{MC}_{mg} and \mathcal{MC}_{ml} satisfy the detailed balance equations in (6.64), as it will be shown in the following sections. The proofs are based on the following result.

Let $p(x, y)$ be a two dimensional probability, and \mathcal{K} be a Markov kernel sampling its conditional probability $p(x|y)$ (or $p(y|x)$). Thus it observes the detailed balance equation,

$$p(x|y)\mathcal{K}(x, x') = p(x'|y)\mathcal{K}(x', x), \quad \forall x, x'. \quad (6.66)$$

Theorem 6.12. *In the above notation, \mathcal{K} observes the general detailed balance equations after augmenting y*

$$p(x, y)\mathcal{K}((x, y), (x', y')) = p(x', y')\mathcal{K}((x', y'), (x, y)).$$

Proof. If $y = y'$, then it is straightforward. If $y \neq y'$ then $\mathcal{K}((x, y), (x', y')) = \mathcal{K}((x', y'), (x, y)) = 0$ because there is no way to go from state (x, y) to state (x', y') .

The conclusion of this theorem is that an algorithm which is reversible when sampling from a conditional probability is also reversible for sampling the full probability.

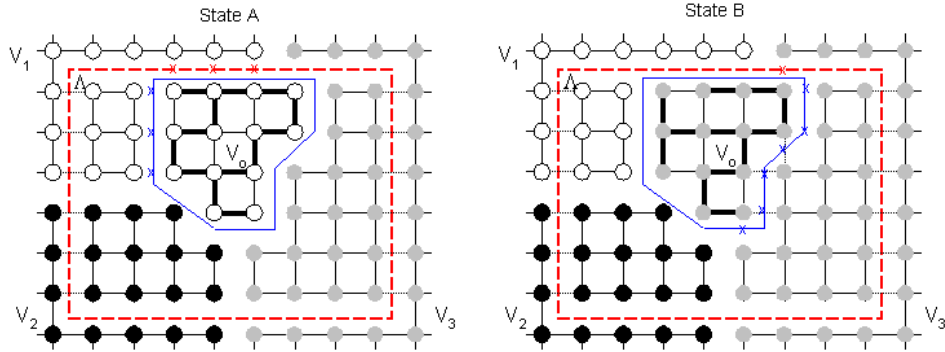


Figure 6.15: Multigrid SW-cut: run SW-cut within an “attention” window Λ with the rest of the labels fixed, and it realizes a reversible move between two states π_A and π_B by flipping the label of $V_o \subset V_\Lambda$.

6.7.1 SW-cuts at multigrid

We first study the multigrid SW-cut. Recall that in each step the SW-cut turns off, probabilistically, the edges in the entire adjacency graph, and this could be less effective especially when G is very large. The concept of multigrid SW-cut is to allow us to select certain “attentional” windows and run the SW-cut within the window. Thus it provides flexibility in designing a “visiting scheme” by

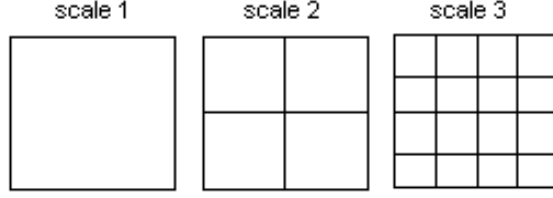


Figure 6.16: Selecting windows in a multigrid scheme

selecting windows of various sizes and locations over time. For example, Fig. 6.16 shows windows in a multigrid arrangement.

Let $G = \langle V, E \rangle$ be the adjacency graph, $\pi = \{V_1, \dots, V_n\}$ the current partition, and Λ an attentional window of arbitrary size and shape. Λ divides the vertices into two subsets $V = V_\Lambda \cup V_{\bar{\Lambda}}$ for vertices inside and outside the window respectively. For example, Fig. 6.15 displays a rectangular window Λ (in red) in a lattice G .

The Λ further removes some edges within each subset $V_i, i = 1, 2, \dots, n$, and we denote them by,

$$\mathcal{C}(V_i|\Lambda) = \{e = \langle s, t \rangle : s \in V_i \cap V_\Lambda, t \in V_i \cap V_{\bar{\Lambda}}\}.$$

For example, in Fig. 6.15 the window Λ intersects with three subsets V_1 (white), V_2 (black), and V_3 (grey), and all edges crossing the (red) rectangle window are removed.

We divide the labeling (coloring or partition) of the vertices V into two parts

$$\pi(V) = (\pi(V_\Lambda), \pi(V_{\bar{\Lambda}})) \quad (6.67)$$

We fix $\pi(V_{\bar{\Lambda}})$ as boundary condition, and sample the labels of vertices within the window by SW-cut.

To summarize, the multigrid SW-cut iterates the following three steps

1. it selects a window Λ of certain size and shape following a probability $\Lambda \sim q(\Lambda)$.
2. For any edges within each subgraph inside the window,
$$e = \langle s, t \rangle, s, t \in \Lambda, \ell_s = \ell_t,$$
it turns off edge e with probability q_e . Thus it obtains a set of connected components $\text{CP}(V_\Lambda)$.
3. It selects $V_o \in \text{CP}(V_\Lambda)$ as a connected component and flips its label according to probability

$$Q(\ell_{\text{new}}(V_o) = j | V_o, \pi) = \frac{1}{C} \prod_{e \in \mathcal{C}_j} q_e \cdot p(\pi_j^*), \quad \forall j \quad (6.68)$$

where π_j^* is the partition by assigning V_o to label j , and $\mathcal{C}_j = \mathcal{C}(V_o, V_j) - \mathcal{C}(V_j|\Lambda)$.

For example, Fig. 6.15 illustrates a reversible move by flipping a connected component V_o (within the blue polygon) between two states π_A and π_B . \mathcal{C}_1 and \mathcal{C}_3 are shown by the blue crosses which are removed by the random procedure.

Following the same procedure as in the previous SW-cut, we can derive the proposal probability ratio for selecting V_o in the two states within Λ .

Theorem 6.13. *The probability ratio for proposing V_o as the candidate subgraph within window Λ at two states π_A and π_B is*

$$\frac{Q(V_o|\pi_A, \Lambda)}{Q(V_o|\pi_B, \Lambda)} = \frac{\prod_{e \in \mathcal{C}(V_o, V_1) - \mathcal{C}(V_1|\Lambda)} q_e}{\prod_{e \in \mathcal{C}(V_o, V_3) - \mathcal{C}(V_3|\Lambda)} q_e}.$$

The difference between this ratio and the ratio in theorem 6.8 is that some edges (see the red crosses in Fig.6.15) no longer participate the computation.

Following the probability in eqn.(6.68) for the new labels, we can prove that it simulates the conditional probability,

$$\pi(V_\Lambda) \sim p(\pi(V_\Lambda)|\pi(V_{\bar{\Lambda}})).$$

Theorem 6.14. *The multigrid SW-cut within window Λ simulates a Markov kernel*

$$\begin{aligned} \mathcal{K}(\Lambda) &= \mathcal{K}(\pi(V_\Lambda), \pi'(V_\Lambda)|\pi(V_{\bar{\Lambda}})), \\ p(\pi(V_\Lambda)|\pi(V_{\bar{\Lambda}}))\mathcal{K}(\pi, \pi') &= p(\pi'(V_\Lambda)|\pi(V_{\bar{\Lambda}}))\mathcal{K}(\pi', \pi). \end{aligned} \quad (6.69)$$

Following theorem 6.12, we have $\mathcal{K}(\Lambda)$ satisfies the general detailed balance equation in eqn.(6.64).

6.7.2 SW-cuts at multi-level

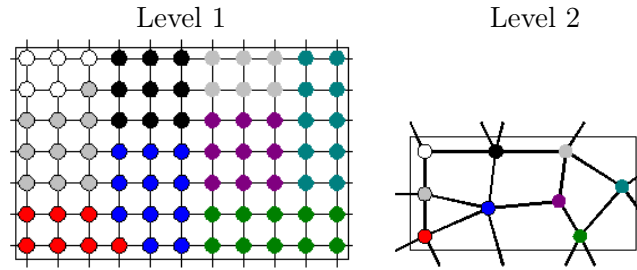


Figure 6.17: Multi-level SW-cut: run SW-cut at two levels.

Now we add a multi-level SW-cut mechanism. Suppose at state $\pi = \{V_1, V_2, \dots, V_n\}$, we “freeze” some subsets $A_k, k \in \{1, \dots, m\}$ such that for any $k, A_k \subset V_i$ for some i . This way, the vertices in each A_k are locked to have the same label. The subsets A_k can represent an intermediary segmentation. For example, for motion segmentation, it is useful to get an intensity segmentation A and group the intensity regions A_k into coherently moving objects.

Thus $G = G^{(1)}$ is reduced to a smaller adjacency graph $G^{(2)} = \langle U, F \rangle$. U is the set of vertices

$$U = \{u_1, \dots, u_m\}, \quad u_k = A_k, k = 1, 2, \dots, m.$$

F is the adjacency relation between the subsets A_k in G .

$$F = \{f = \langle u_i, u_j \rangle : \mathcal{C}(A_i, A_j) \neq \emptyset\}.$$

Fig.6.17 illustrates an example with $m = 9$. We run the SW-cut on level 2 based on new discriminative heuristics $q^{(2)}$ which measure the similarity of A_i, A_j , $q^{(2)}(\pi(U)) = \prod_{f \in F} q_f^{(2)}$. In general, these heuristics are more informative than the lower level, so the SW-cuts moves are more meaningful and the convergence is faster.

The partition space for graph $G^{(2)}$ is a projection of Ω_π ,

$$\Omega_\pi(G^{(2)}) = \{\pi : \ell(s) = \ell(t), \quad \forall s, t \in A_i, \quad i = 1, 2, \dots, m.\}$$

Obviously, the SW-cut on level 2 simulates a Markov chain with kernel $\mathcal{K}^{(2)}$ which has invariant probability $p(\pi(U)|A)$, the probability of $p(\pi)$ conditional on the relations $\ell(s) = \ell(u_i)$ for all $s \in A_i$ and all i .

Following theorem 6.12, we have that $\mathcal{K}^{(2)}$ satisfies the general detailed balance equation (6.64).

Summary. Suppose we design a visiting scheme for selecting the windows $\Lambda \sim q_w(\Lambda)$ and level $\sigma \sim q_l(\sigma)$ over time. Then the generalized SW-cut has a mixed Markov kernel

$$\mathcal{K} = \sum_{\sigma} \sum_{\Lambda} q_l(\sigma) q_w(\Lambda) \mathcal{K}^{(\sigma)}(\Lambda).$$

As each $\mathcal{K}^{(\sigma)}(\Lambda)$ observes the detailed balance equations, so is \mathcal{K} . When the windows cover the entire graph, it is also irreducible and its states follows $p(\pi)$ at convergence.

6.8 Subspace Clustering

Subspace clustering is the problem of grouping an unlabeled set of points into a number of clusters corresponding to subspaces of the ambient space. This problem has applications in unsupervised learning and computer vision. One of the computer vision applications is sparse motion segmentation, where a number of feature point trajectories need to be grouped into a small number of clusters according to their common motion model. The feature point trajectories are obtained by detecting a number of feature points using an interest point detector and tracking them through many frames using a feature point tracker or an optical flow algorithm.

A common approach in the state of the art sparse motion segmentation methods [60] [111] [121] [198] [210] is to project the feature trajectories to a lower dimensional space and use a subspace clustering method based on spectral clustering to group the projected points and obtain the motion segmentation.

Even though these methods obtain very good results on standard benchmark datasets, the spectral clustering algorithm requires expensive computation of eigenvectors and eigenvalues on an $N \times N$ dense matrix where N is the number of data points. In this manner, the computation time for these subspace clustering/motion segmentation methods scales as $O(N^3)$, so it can become prohibitive for large problems (e.g. $N = 10^5 - 10^6$).

This section presents a completely different approach to subspace clustering, based on the Swendsen-Wang Cut (SWC) algorithm [9]. The subspace clustering problem is formulated as Maximum A Posteriori (MAP) optimization problem in a Bayesian framework with Ising/Potts prior [165] and likelihood based on a linear subspace model. The SWC graph is constructed as a k-NN graph from an affinity matrix.

Overall, the proposed method provides a new perspective to solve the subspace clustering problem, and demonstrates the power of Swendsen-Wang Cuts algorithm in clustering problems.

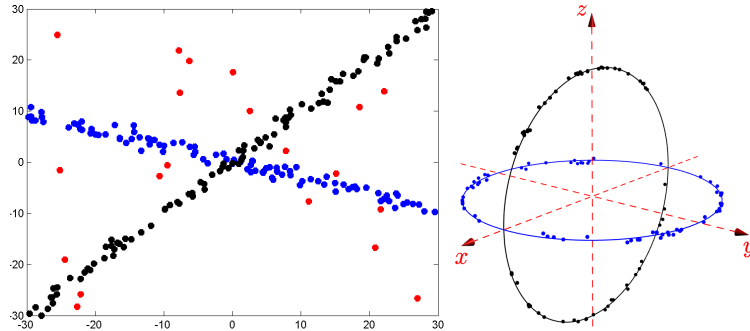


Figure 6.18: Left. two subspaces in 2D. Right. two 2D subspaces in 3D. The points in both 2D subspaces have been normalized to unit length. Due to noise, the points may not lie exactly on the subspace. One can observe that the angular distance finds the correct neighbors in most places except at the plane intersections.

Subspace Clustering by Spectral Clustering

Given a set of points $\{\mathbf{x}_1, \dots, \mathbf{x}_N\} \in \mathbb{R}^D$, the subspace clustering problem is to group the points into a number of clusters corresponding to linear subspaces of \mathbb{R}^D . The problem is illustrated in Figure 6.18, left, showing two linear subspaces and a number of outliers in \mathbb{R}^2 .

A popular subspace clustering method [56] [111] [167] is based on spectral clustering, which relies on an affinity matrix that measures how likely any pair of points belong to the same subspace.

Spectral clustering [150, 175] is a generic clustering method that groups a set of points into clusters based on their connectivity. The point connectivity is given as an $N \times N$ affinity matrix A with A_{ij} close to 1 if point i is close to point j and close to zero if they are far away. The quality of the affinity matrix is very important for obtaining good clustering results. The affinity matrix for spectral subspace clustering is computed as follows..

First, the the points are normalized to unit length [56, 111, 167], as shown in Figure 6.18, right.

Then the following affinity measure based on the angle between the vectors has been proposed in [111]

$$A_{ij} = \left(\frac{x_i^T x_j}{\|x_i\|_2 \|x_j\|_2} \right)^{2\alpha}, \quad (6.70)$$

where α is a tuning parameter, and the value $\alpha = 4$ has been used in [111].

Fig 6.18, right shows two linear subspaces, where all points have been normalized. It is intuitive to find that the points tend to lie in the same subspace as their neighbors in angular distance except those near the intersection of the subspaces.

6.8.1 Subspace Clustering by Swendsen-Wang Cuts

This section presents a novel subspace clustering algorithm that formulates the subspace clustering problem as a MAP estimation of a posterior probability in a Bayesian framework and uses the Swendsen-Wang Cuts algorithm [9] for sampling and optimization.

A subspace clustering solution can be represented as a partition (labeling) $\pi : \{1, \dots, N\} \rightarrow \{1, \dots, M\}$ of the input points $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^D$. The number $M \leq N$ is the maximum number of allowed clusters.

In this section is assumed that an affinity matrix A is given, representing the likelihood for any pair of points to belong to the same subspace. One form of A has been given in (6.70) and another one will be given below.

Posterior Probability

A posterior probability will be used to evaluate the quality of any partition π . A good partition can then be obtained by maximizing the posterior probability in the space of all possible partitions. The posterior probability is defined in a Bayesian framework

$$p(\pi) \propto \exp[-E_{data}(\pi) - E_{prior}(\pi)].$$

The normalizing constant is irrelevant in the optimization since it will cancel out in the acceptance probability.

The data term $E_{data}(\pi)$ is based on the fact that the subspaces are assumed to be linear. Given the current partition (labeling) π , for each label l an affine subspace L_l is fitted in a least squares sense through all points with label l . Denote the distance of a point x with label l to the linear space L_l as $d(x, L_l)$. Then the data term is

$$E_{data}(\pi) = \sum_{l=1}^M \sum_{i, \pi(i)=l} d(x_i, L_l) \quad (6.71)$$

The prior term $E_{prior}(\pi)$ is set to encourage tightly connected points to stay in the same cluster.

$$E_{prior}(\pi) = -\rho \sum_{\langle i,j \rangle \in E, \pi(i) \neq \pi(j)} \log(1 - A_{ij}), \quad (6.72)$$

where ρ is a parameter controlling the strength of the prior term. It will be clear in the next section that this prior is exactly the Potts model (6.1) that would have A_{ij} as the edge weights in the original SW algorithm.

The SWC algorithm could automatically decide the number of clusters, however in this chapter, as in most motion segmentation algorithms, it is assumed that the number of subspaces M is known. Thus the new label for the component V_0 is sampled with uniform probability from the number M of subspaces:

$$q(c_{V_0} = l' | V_0, \pi) = 1/M.$$

Graph Construction

Section 6.8 described a popular subspace clustering method based on spectral clustering. Spectral clustering optimizes an approximation of the normalized cut or the ratio cut [200], which are discriminative measures. In contrast, the proposed subspace clustering approach optimizes a generative model where the likelihood is based on the assumption that the subspaces are linear. It is possible that the discriminative measures are more flexible and work better when the linearity assumption is violated, and will be studied in future work.

The following affinity measure, inspired by [111], will be used in this work

$$A_{ij} = \exp(-m \frac{\theta_{ij}}{\bar{\theta}}), \quad i \neq j \quad (6.73)$$

where θ_{ij} is based on the angle between the vectors x_i and x_j ,

$$\theta_{ij} = 1 - \left(\frac{x_i^T x_j}{\|x_i\|_2 \|x_j\|_2} \right)^2,$$

and $\bar{\theta}$ is the average of all θ . The parameter m is a tuning parameter to control the size of the connected components obtained by the SWC algorithm. The subspace clustering performance with respect to this parameter will be evaluated in Section 6.8.2 for motion segmentation.

The affinity measure based on the angular information between points enables to obtain the neighborhood graph, for example based on the k -nearest neighbors. After the graph has been obtained, the affinity measure is also used to obtain the edge weights for making the data driven clustering proposals in the SWC algorithm as well as for the prior term of the posterior probability.

The graph $G = (V, E)$ has as vertices the set of points that need to be clustered. The edges E are constructed based on the proposed distance measure from eq. (6.73). Since the distance measure is more accurate in finding the nearest neighbors (NN) from the same subspace, the graph is constructed as the k -nearest neighbor graph (kNN), where k is a given parameter.

Examples of obtained graphs will be given in Section 6.8.2.

Optimization by Simulated Annealing

The SWC algorithm is designed for sampling the posterior probability $p(\pi)$. To use SWC for optimization, a simulated annealing scheme should be applied while running the SWC algorithm.

Simulated annealing means the probability used by the algorithm is not $p(\pi)$ but $p(\pi)^{1/T}$ where T is a "temperature" parameter that is large at the beginning of the optimization and is slowly

```

Input:  $N$  points  $(\mathbf{x}_1, \dots, \mathbf{x}_N)$  from  $M$  subspaces
Construct the adjacency graph  $G$  as a  $k$ -NN graph using eq (6.73).
for  $r = 1, \dots, Q$  do
  Initialize the partition  $\pi$  as  $\pi(i) = 1, \forall i$ .
  for  $i = 1, \dots, N^{it}$  do
    1. Compute the temperature  $T_i$  using eq (6.74).
    2. Run one step of the SWC algorithm using  $p(\pi|I) = p^{1/T_i}(\pi)$  in eq (6.46).
  end for
  Record the clustering result  $\pi_r$  and the final probability  $p_r = p(\pi_r)$ .
end for
Output: Clustering result  $\pi_r$  with the largest  $p_r$ .

```

Figure 6.19: The Swendsen-Wang Cuts algorithm for subspace clustering.

decreased according to an annealing schedule. If the annealing schedule is slow enough, it is theoretically guaranteed [101] that the global optimum of the probability $p(\pi)$ will be found.

In reality we use a faster annealing schedule, and the final partition π will only be a local optimum. We use an annealing schedule that is controlled by three parameters: the start temperature T_{start} , the end temperature as T_{end} , and the number of iterations N^{it} . The temperature at step i is calculated as

$$T_i = \frac{T_{\text{end}}}{\log \left(\frac{i}{N} [e - \exp(\frac{T_{\text{end}}}{T_{\text{start}}})] + \exp(\frac{T_{\text{end}}}{T_{\text{start}}}) \right)}, i = 1, N^{it} \quad (6.74)$$

To better explore the probability space, we also use multiple runs with different random initializations. Then the final algorithm is shown in Figure 6.19.

Complexity Analysis

Let N be the number of points in \mathbb{R}^D that need to be clustered. The computation complexity of the SWC subspace clustering method can be broken down as follows:

- The adjacency graph construction is $O(N^2 D \log k)$ where D is the space dimension. This is because one needs to calculate the distance from each point to the other $N - 1$ points and use a heap to maintain its k -NNs.
- Each of the N^{it} iterations of the SWC algorithm is $O(N\alpha(N))$, as discussed in section 6.4.4. Computing $E_{\text{data}}(\pi)$ involves fitting linear subspaces for each motion cluster, which is $O(D^2 N + D^3)$, while computing the $E_{\text{prior}}(\pi)$ is $O(N)$. The number of iterations is $N^{it} = 2000$, so all the SWC iterations take $O(N\alpha(N))$ time.

In conclusion, the entire algorithm complexity in terms of the number N of points is $O(N^2)$ so it scales better than spectral clustering for large problems.

6.8.2 Application: Sparse Motion Segmentation

This section presents an application of the proposed subspace clustering algorithm to motion segmentation.

Most recent works on motion segmentation use the affine camera model, which is approximatively satisfied when the objects are far from the camera. Under the affine camera model, a point on the image plane (x, y) is related to the real world 3D point X by

$$\begin{bmatrix} x \\ y \end{bmatrix} = A \begin{bmatrix} X \\ 1 \end{bmatrix}, \quad (6.75)$$

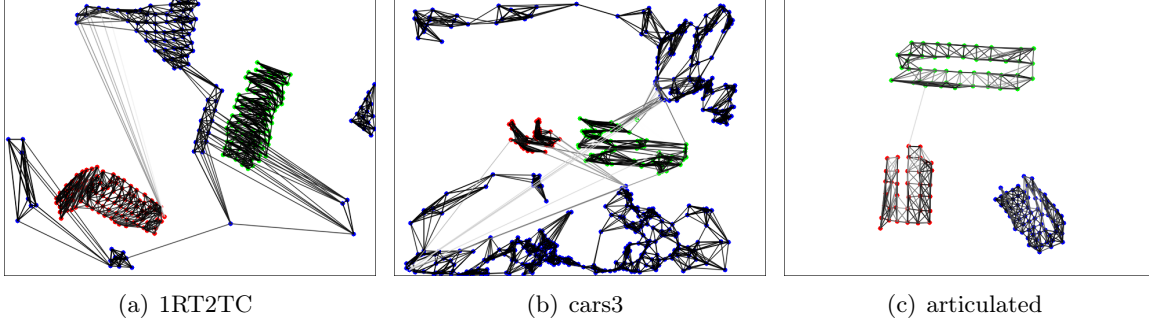


Figure 6.20: Examples of SWC weighted graphs for a checkerboard (left), traffic (mid) and articulated (right) sequence. Shown are the feature point positions in the first frame. The edge intensities represent their weights from 0 (white) to 1 (black).

where $A \in \mathbb{R}^{2 \times 4}$ is the affine motion matrix.

Let $t_i = (x_i^1, y_i^1, x_i^2, y_i^2, \dots, x_i^F, y_i^F)^T, i = 1, \dots, N$ be the trajectories of tracked feature points in F frames (2D images), where N is the number of trajectories. Let the *measurement matrix* $W = [t_1, t_2, \dots, t_N]$ be constructed by assembling the trajectories as columns.

If all trajectories undergo the same rigid motion, equation (6.75) implies that W can be decomposed into a *motion matrix* $M \in \mathbb{R}^{2F \times 4}$ and a *structure matrix* $S \in \mathbb{R}^{4 \times N}$ as

$$W = MS$$

$$\begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_N^1 \\ y_1^1 & y_2^1 & \cdots & y_N^1 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^F & x_2^F & \cdots & x_N^F \\ y_1^F & y_2^F & \cdots & y_N^F \end{bmatrix} = \begin{bmatrix} A^1 \\ \vdots \\ A^F \end{bmatrix} \begin{bmatrix} X_1 & \cdots & X_N \\ 1 & \cdots & 1 \end{bmatrix}$$

where A^f is the affine object to world transformation matrix at frame f . It implies that $\text{rank}(W) \leq 4$. Since the entries of the last row of S are always 1, under the affine camera model, the trajectories of feature points from a rigidly moving object reside in an affine subspace of dimension at most 3.

In general, we are given a measurement matrix W that contains trajectories from multiple possibly nonrigid motions. The task of motion segmentation is to cluster together all trajectories coming from each motion.

A popular approach [56] [111] [167] [198] is to project the trajectories to a lower dimensional space and to perform subspace clustering in that space, using spectral clustering as described in section 6.8. These methods differ in the projection dimension D and in the affinity measure A used for spectral clustering.

Dimension Reduction

Dimension reduction is an essential preprocessing step for obtaining a good motion segmentation. To realize this goal, the truncated SVD is often applied [56, 111, 167, 198].

To project the measurement matrix $W \in \mathbb{R}^{2F \times N}$ to $X = [x_1, \dots, x_N] \in \mathbb{R}^{D \times N}$, where D is the desired projection dimension, the matrix W is decomposed via SVD as $W = U\Sigma V^T$ and the first D columns of the matrix V are chosen as X^T .

The value of D for dimension reduction is also a major concern in motion segmentation. This value has a large impact on the speed and accuracy of the final result, so it is very important to select the best dimension to perform the segmentation. The dimension of a motion is not fixed, but



Figure 6.21: Sample images from some sequences of three categories in the Hopkins 155 database with ground truth superimposed.

can vary from sequence to sequence, and since it is hard to determine the actual dimension of the mixed space when multiple motions are present, different methods may have different dimensions for projection.

The GPCA [198] suggests to project the trajectories onto a 5-dimensional space. ALC [167] chooses to use the sparsity-preserving dimension $d_{sp} = \operatorname{argmin}_{d \geq 2D \log(2T/d)} d$ for D -dimensional subspaces. The SSC [60] and LRR [121] simply projects the trajectories to the $4M$ subspace, where M is the number of motions. Some methods [56, 111] use an exhaustive search strategy to perform the segmentation in spaces with a range of possible dimensions and pick the best result. In this chapter, we find that projecting to dimension $D = 2M + 1$ can generate good results.

The computation complexity of computing the SVD of a $m \times n$ matrix U when $m \gg n$ is $O(mn^2 + n^3)$ [185]. If $n \gg m$ then it is faster to compute the SVD of U^T , which takes $O(nm^2 + m^3)$.

Assuming that $2F \ll N$, it means that the SVD of W can be computed in $O(NF^2 + F^3)$ operations.

After projecting to the subspace of dimension $D = 2M + 1$, the SWC subspace clustering algorithm from Section 6.8.1 is applied and the clustering result gives the final motion segmentation result.

Experiments on the Hopkins 155 Dataset

This section presents experiments with the SWC-based motion segmentation algorithm on the Hopkins 155 motion database [187]. The database consists of 155 sequences of two and three motions. The ground-truth segmentation is also provided for evaluation purposes. Based on the content of the video, the sequences could be categorized into three main categories: checkerboard, traffic, and articulated sequences, with examples shown in Figure 6.21. The trajectories are extracted automatically by a tracker, so they are slightly corrupted by noise.

As already mentioned, before applying the SWC algorithm, the dimension of the data is reduced from $2F$ to $D = 2M + 1$, where M is the number of motions. After the projection, the initial labeling state in the SWC algorithm has all points having the same label.

The motion segmentation results are evaluated using the misclassification error rate

$$\text{Misclassification Rate} = \frac{\# \text{ of misclassified points}}{\text{total } \# \text{ of points}} \quad (6.76)$$

Parameter settings. The motion segmentation algorithm has a number of tuning parameters that were held constant to the following values. The number of NN (nearest neighbors) for graph construction is $k = 7$, the parameter m in the affinity measure (6.73) is $m = 10$, and the prior coefficient in (6.72) is $\rho = 2.2$. The annealing parameters are $T_{\text{start}} = 1$, $T_{\text{end}} = 0.01$, $N^{it} = 2000$.

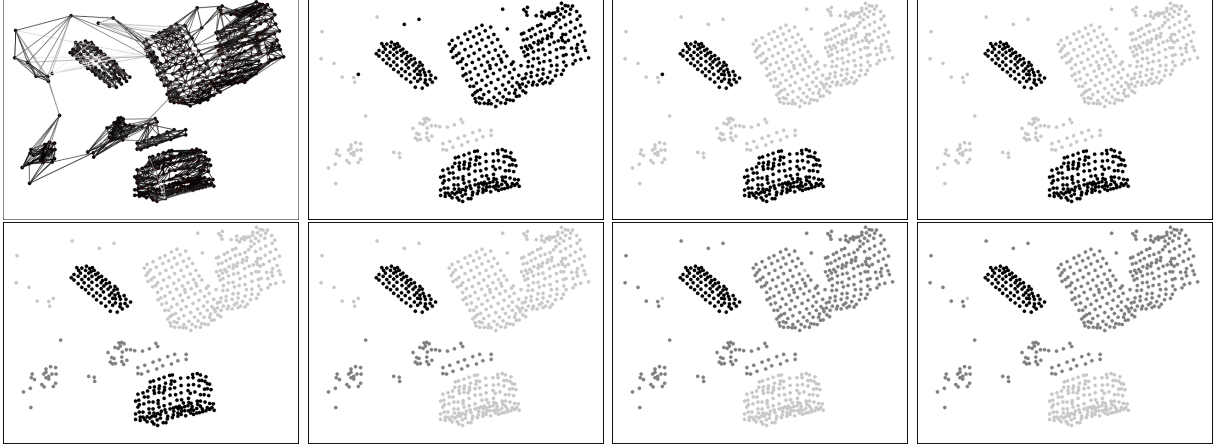


Figure 6.22: SWC clustering of the Hopkins 155 sequence 1R2TCR, containing $M = 3$ motions. Shown are the feature point positions in the first frame, having colors as the labeling states π obtained while running the SWC algorithm from the initial state (top left) to the final state (bottom right).

The number of independent runs to obtain the most probable partition is $Q = 10$. An example of all the partition states during an SWC run is shown in Figure 6.22.

Results. The average and median misclassification errors are listed in Table 6.1. For accuracy, the results of the SWC algorithm from Table 6.1 are averaged over 10 runs and the standard deviations are shown in parentheses. In order to compare the SWC method with the state of the art methods, we also list the results of ALC [167], SC [111], SSC [60] and VC [56].

Table 6.1: Misclassification rates (in percent) of different motion segmentation algorithms on the Hopkins 155 dataset.

Method	ALC	SC	SSC	VC	SWC (std)	SC ⁴	SC ^{4k}	KSP
All (2 motion)								
Average	2.40	0.94	0.82	0.96	1.49 (0.19)	11.50	7.82	4.76
Median	0.43	0.00	0.00	0.00	0.00 (0.00)	2.09	0.27	0.00
All (3 motion)								
Average	6.69	2.11	2.45	1.10	2.62 (0.13)	19.55	11.25	9.00
Median	0.67	0.37	0.20	0.22	0.81 (0.00)	18.88	1.42	1.70
All sequences combined								
Average	3.37	1.20	1.24	0.99	1.75 (0.15)	13.32	8.59	5.72
Median	0.49	0.00	0.00	0.00	0.00 (0.00)	6.46	0.36	0.31

The SWC based algorithm obtains average errors that are less than twice the errors of the other methods. In our experiments we observed that the energy of the final state is usually smaller than the energy of the ground truth state. This fact indicates that the SWC algorithm is doing a good job optimizing the model but the Bayesian model is not accurate enough in its current form and needs to be improved.

Also shown in Table 6.1 are the columns labeled SC⁴ and SC^{4k} representing the misclassification errors of the SC method [111] with an affinity matrix with 4-NN and 4k-NN respectively. These errors are 13.32 and 8.59 respectively and indicate that the Spectral Clustering really needs a dense affinity matrix to work well, and it cannot be accelerated using sparse matrix operations.

Finally, the performance of the SWC-based algorithm is compared with the KASP algorithm [209], which is a fast approximate spectral clustering and was used in place of the spectral clustering step in the SC method [111]. The data reduction parameter used was $\gamma = 10$, which still results in a $O(N^3)$ clustering algorithm. The total misclassification error is 5.72, about three times larger than the SWC method.

Scalability Experiments on Large Data



Figure 6.23: Selected frames of sequence cars10 with 1000 tracked feature points.

In order to evaluate the scalability of different algorithms, sequences with a large number of trajectories are needed. The trajectories can be generated by some optical flow algorithm, but it is difficult to obtain the ground truth segmentation and remove bad trajectories caused by occlusions. Brox et.al. [28] provided a dense segmentation for some frames in 12 sequences in the Hopkins 155 dataset¹. From them, we picked the cars10 sequence and tracked all pixels of the first frame using the Classic+NL method [179]. There are two reasons for choosing cars10. First, it has three motions, two moving cars and the background. Second, the two moving cars are relatively large in the video, so that a large number of trajectories can be obtained from each motion.

There are 30 frames in the sequence, and 3 of them have a dense manual segmentation of all pixels. We removed trajectories that have different labels on the 3 ground truth frames. To avoid occlusion, the trajectories close to the motion boundaries were also removed. Plus, we only kept the full trajectories for clustering. Finally, we obtained around 48,000 trajectories as a pool. From the pool, different numbers N of trajectories were subsampled for evaluation. For each given N , a total of N trajectories were randomly selected from the pool such that the number of trajectories in each of the three motions was roughly the same. For example, to generate $N = 1000$ trajectories, we would randomly pick from the pool 333 trajectories from two of the motions and 334 trajectories from the third motion. If there were not enough trajectories available from one motion, we added more from the motion that has the most trajectories.

We compared the SWC method with the SC algorithm [111] discussed in Section 6.8, which is one of the fastest and most accurate algorithms [56] based on spectral clustering. We generated data containing between 1,000 to 15,000 trajectories, and applied the two segmentation algorithms. Sample frames are shown in Figure 6.23. The parameters for SC were kept the same as in the original paper, and those for SWC were identical with the experiments from Table 6.1. The SC algorithm is implemented in Matlab (which has optimized SVD algorithms), while the SWC code is in C++. The experiments were performed on a Windows machine with an Intel core i7-3970 CPU and 12 GB memory. We also generated data with $N = 24,000$ and $N = 48,000$ trajectories for SWC clustering. For SC, the same experiments could not be conducted because Matlab ran out of

¹<http://lmb.informatik.uni-freiburg.de/resources/datasets/>

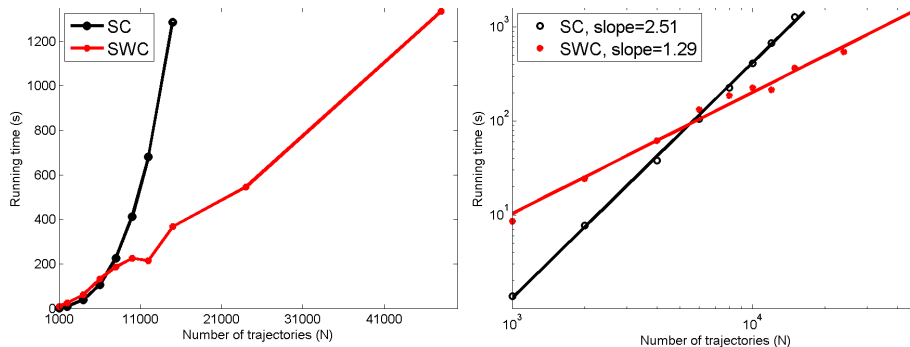


Figure 6.24: Left. Computation time (sec) vs number of trajectories N for SC and SWC. Right: log-log plot of same data with the fitted regression lines.

memory.

Table 6.2: Average misclassification rate for the sequence cars10 (in percent).

Number of Trajectories N	SC	SWC
1000 to 15,000	2.77	0.99
24,000 to 48,000	-	1.00

The misclassification rate is recorded in Table 6.2 and the running time is shown on Figure 6.24. Table 6.2 shows that both methods perform well and the misclassification rate of SWC is about one third of that of the SC.

From Figure 6.24, which shows the computation time vs the number N of trajectories, one could find that for a small number of trajectories, the SC is faster than SWC, but for more than $N = 6,000$ trajectories, the computation time of SC is greater than that of SWC, and increases much faster. We also plot the $\log(\text{time})$ vs. $\log(N)$ and use linear regression to fit lines through the data points of the two methods. If the slope of the line is α , then the computation complexity is $O(N^\alpha)$. We observe that the slope of SC is 2.52 while the slope for SWC is 1.29, which is consistent with the complexity analysis of Section 6.8.1.

6.9 C4: Clustering Cooperative and Competitive Constraints

Many vision tasks, such as scene labeling [109, 162, 170], object detection/recognition [62, 184], segmentation [43, 193], and graph matching [38, 120] are formulated as energy minimization (or maximum a posteriori probability) problems defined on graphical models – Markov random fields [17, 75], conditional random fields [109, 110], or hierarchical graphs [73, 221]. These optimization problems become exceedingly difficult when there are multiple solutions, i.e. distinct modes with high probabilities and, in some cases, equal probability.

Fig. 6.25 shows examples of typical scenarios that have multiple, equally likely solutions in the absence of further context. The top row shows the well-known Necker Cube which has two valid 3D interpretations. The middle row is the Wittgenstein illusion, in which the drawing can appear to be either a duck or a rabbit. Without further context, we cannot determine the correct labeling. The bottom row shows an aerial image for scene labeling. It can be explained as either a roof with vents or a parking lot containing cars.

Computing multiple solutions is important for preserving the intrinsic ambiguities and avoiding early commitment to a single solution which, even if it's currently the globally optimal one, may

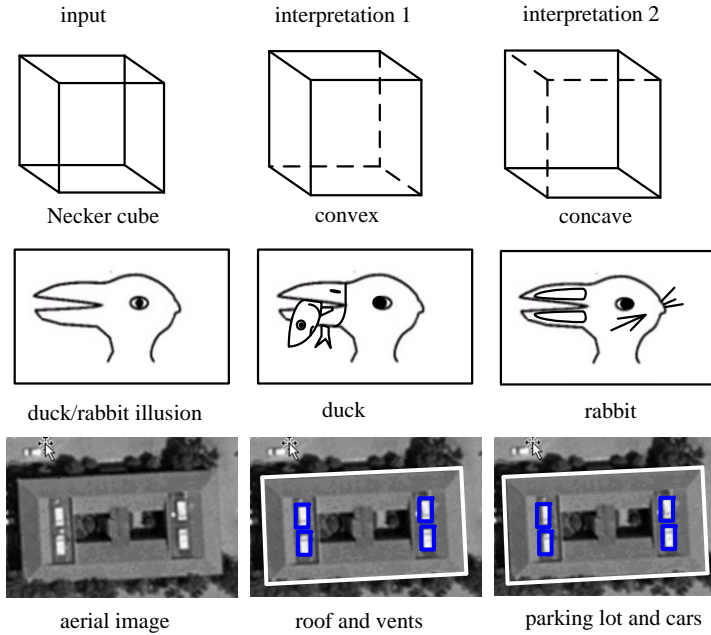


Figure 6.25: Problems with multiple solutions: (top) the Necker Cube; (Middle) the Wittgenstein illusion; and (Bottom) An aerial image interpreted as either a roof with vents or a parking lot with cars. Ambiguities should be preserved until further context arrives.

turn out to be less favorable when later context arrives. However, it is a persistent challenge to enable algorithms to climb out of local optima and to jump between solutions far apart in the state space. Popular energy minimization algorithms, such as Iterative Conditional Modes (ICM) [17], Loopy Belief Propagation (LBP) [108, 204], and graph cuts [23, 105] compute one solution and thus do not address this problem. Existing MCMC algorithms, such as various Gibbs samplers [75, 124], DDMCMC [193], and Swendsen-Wang cuts [9, 180], promise global optimization and ergodicity in the state space, but often need long waiting time in moving between distinct modes, which needs a sequence of lucky moves up the energy landscape before it goes down.

In this section, our objective is to develop an algorithm that can discover multiple solutions by jumping out of equal probability states and thus preserve the ambiguities on rather general settings:

1. The graph can be flat, such as a MRF or CRF, or hierarchical, such as a parse graph.
2. The graph may have positive (cooperative) and negative (competitive or conflicting) edges for both hard or soft constraints.
3. The probability (energy) defined on the graph is quite general, even with energy terms involving more than two nodes.

In vision, it is safe to assume that the graph is locally connected and we do not consider the worst case scenario where graphs are fully connected.

In the 1970s, many problems, including line drawing interpretation and scene labeling, were posed as constraint satisfaction problems (CSPs). The CSPs were either solved by heuristic search methods [158] or constraint propagation methods [6, 129]. The former keeps a list of open nodes for plausible alternatives and can backtrack to explore multiple solutions. However, the open list can

become too long to maintain when the graph is large. The latter iteratively updates the labels of nodes based on their neighbors. One well-known constraint propagation algorithm is the relaxation labeling method by Rosenfeld, Hummel, and Zucker in 1976 [170].

In the 1980s, the famous Gibbs sampler – a probabilistic version of relaxation labeling – was presented by Geman and Geman in 1984 [75]. The update of labels is justified in a solid MCMC and MRF framework and thus is guaranteed to sample from the posterior probabilities. In special cases, the Gibbs sampler is equal to belief propagation [158] for polytrees and to dynamic programming in chains. The Gibbs sampler is found to slow down critically when a number of nodes in the graph are strongly coupled.

Fig. 6.26 illustrates an example of the difficulty with strongly coupled graphs using the Necker Cube. The six internal lines of the figure are divided into two coupling groups: (1-2-3) and (4-5-6). Lines in each group must have the same label (concave or convex) to be valid as they share the two 'Y'-junctions. Thus, updating the label of a single line in a coupled group does not move at all, unless we update the label of the whole group together, i.e. all six labels in one step.

The problem is that we don't know which nodes in the graph are coupled and to what extent they are coupled for general problems with large graphs. In 1987, a breakthrough came from two physicists, Swendsen and Wang [180], who proposed a cluster sampling technique. The Swendsen-Wang (SW) method finds coupled groups, called "clusters", dynamically by turning the edges in the graph on/off according to the probabilities defined on these edges. The edge probability measures the coupling strengths. Unfortunately, their algorithm only works for the Ising and Potts models. We will discuss the SW method in later sections.

There were numerous attempts made to improve MCMC methods in the 1990's (see Liu [123] for surveys), such as the block Gibbs sampler [124]. Green formulated reversible jumps in 1995 [87] following the jump-diffusion algorithm by Grenander and Miller in 1994 [88]. In 1999, Cooper and Frieze analyzed the convergence speed of SW using a path coupling technique and showed that the SW method has a polynomial mixing time when the nodes in the graph are connected to a constant number of neighbors [41]. Nevertheless, it was also shown that SW could mix slowly under conditions when graphs were heavily or fully connected [86].

In the 2000s, a few non-MCMC methods generated remarkable impacts on the vision community. For example, the loopy belief propagation (LBP) algorithm by Weiss et. al. [204] and the graph cut algorithms by Boykov, Kolmogorov, et. al. [23, 105]. These algorithms are very fast and work well on special class of graph structures and energy functions. In addition, techniques such as survey propagation [24] have had great success in statistical physics. In the case of multimodal energy functions, however, it can be difficult for these techniques to converge properly, as we will see.

On the MCMC side, Tu and Zhu developed the Data-Driven Markov Chain Monte Carlo (DDMCMC) algorithm for image segmentation in 2002 [193], which uses bottom-up discriminative probabilities to drive the Markov chain moves. They also developed a "K-adventurer" procedure to keep multiple solutions. The DDMCMC method was also used by Dellaert [153] for tracking bee dances. Dellaert also used MCMC to explore correspondences for structure-from-motion problems, even incorporating a "jump parameter" to allow the algorithm to jump to new solutions [50]. In 2005, Barbu, 2005 generalizing and Zhu proposed the SW-cut algorithm [9] which, for the first time, generalized the SW method to arbitrary probabilities models. As we will discuss in later sections, the SW-cut did not consider negative edges, high order constraints, or hierarchical graphs and is less effective in swapping between competing solutions. The C^4 algorithm in this section is a direct generalization of the SW-cut algorithm [9].

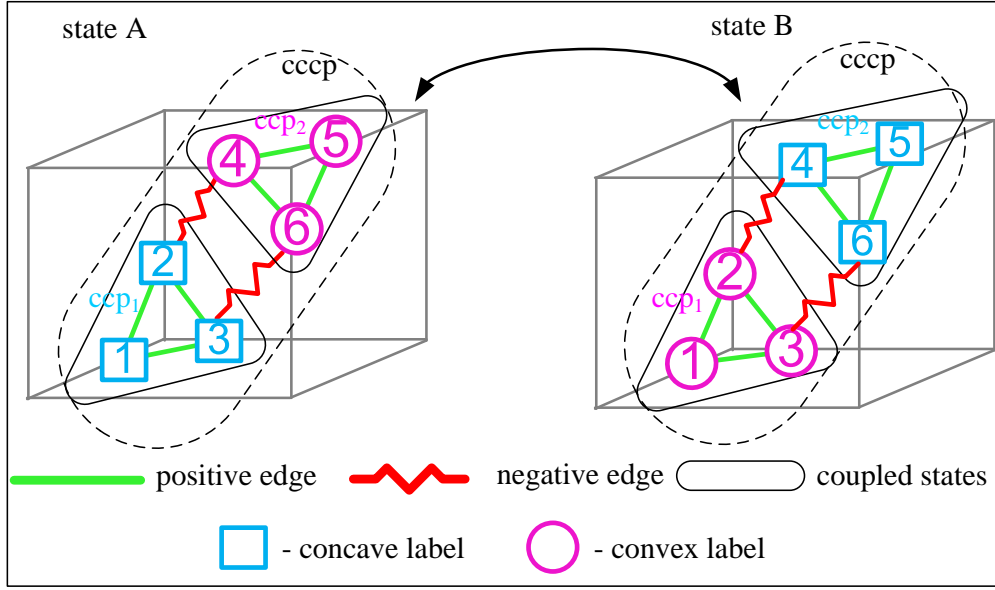


Figure 6.26: Swapping between the two interpretations of the Necker Cube. Locally coupled labels are swapped with alternate labelings to enforce global consistency. See text for explanation.

6.9.1 Overview of the Major Concepts of C^4

In this section we present a probabilistic clustering algorithm called *Clustering Cooperative and Competitive Constraints* (C^4) for computing multiple solutions in graphical models. We consider two types of graphs:

Adjacency graphs treat each node as an entity, such as a pixel, a superpixel, a line, or an object, which has to be labeled in K -classes (or colors). Most MRFs and CRFs used in computer vision are adjacency graphs.

Candidacy graphs treat each node as a candidate or hypothesis, such as a potential label for an entity, or a detected object instance in a window, which has to be confirmed ('on') or rejected ('off'). In other words, the graph is labeled with $K = 2$ colors.

As we will show in section 6.9.2, an adjacency graph can always be converted to a bigger candidacy graph. In both cases, the tasks are posed as graph coloring problems on MRF, CRF or hierarchical graphs. There are two types of edges expressing either hard or soft constraints (or coupling) between the nodes.

Positive edges are cooperative constraints that favor the two nodes having the same label in an adjacency graph or being turned on (or off) simultaneously in a candidacy graph.

Negative edges are competitive or conflicting constraints that require the two nodes to have different labels in an adjacency graph or one node to be turned on and the other turned off in a candidacy graph.

In Fig. 6.26, we show that the Necker cube can be represented in an adjacency graph with each line being a node. The six internal lines are linked by 6 positive edges (in green) and two negative edges (in red and wiggly). Lines 2 and 4 have a negative edge between them as they intersect with each other, as do lines 3 and 6. We omit the labeling of the six outer lines for clarity.

In this section, the edges play computational roles, and are used to dynamically group nodes which are strongly coupled. On each positive or negative edge, we define an *edge probability* (using

bottom-up discriminative models) for the coupling strength. Then we design a *protocol* for turning these edges on and off independently according to their edge probabilities respectively for each iteration. The protocol is common for all problems while the edge probabilities are problem specific. This probabilistic procedure turns off some edges, and all the edges that remain 'on' partition the graph into some connected components (*ccp*'s).

A *ccp* is a set of nodes that are connected by the positive edges. For example, Fig. 6.26 has two *ccp*'s: *ccp*₁ includes nodes 1-2-3 and *ccp*₂ includes nodes 4-5-6. Each *ccp* is a locally coupled sub-solution.

A *cccp* is a composite connected component that consists of a number of *ccp*'s connected by negative edges. For example, Fig. 6.26 has one *cccp* containing *ccp*₁ and *ccp*₂. Each *cccp* contains some conflicting sub-solutions.

At each iteration, C^4 selects a *cccp* and updates the labels of all nodes in the *cccp* simultaneously so that (i) nodes in each *ccp* keep the same label to satisfy the positive or coupling constraints, and (ii) different *ccp*'s in the *cccp* are assigned different labels to observe the negative constraints.

Since C^4 can update a large number of nodes in a single step, it can move out of local modes and jump effectively between multiple solutions. The protocol design groups the *cccp*'s dynamically and guarantees that each step follows the MCMC requirements, such as detailed balance equations and thus it samples from the posterior probability.

We evaluate C^4 against other popular algorithms in the literature by two criteria.

1. The speed that they converge to solutions. In some studied cases, we know the global minimum solutions.
2. The number of unique solution states generated by the algorithms over time. This measures how "dynamic" an algorithm is.
3. The estimated marginal probability at each site in the graphical model after convergence.

6.9.2 Graphs, Coupling, and Clustering

Adjacency and candidacy graphs

We start with a flat graph G that we will extend to a hierarchical graph in section 6.9.6,

$$G = \langle V, E \rangle, \quad E = E^+ \cup E^-. \quad (6.77)$$

Here $V = \{v_i, i = 1, 2, \dots, n\}$ is a set of vertices or nodes on which variables $X = (x_1, \dots, x_n)$ are defined, and $E = \{e_{ij} = (v_i, v_j)\}$ is a set of edges which is divided into E^+ and E^- for positive (cooperative) and negative (competitive or conflicting) constraints respectively. We consider two types of graphs for G :

Adjacency graph, where each node $v_i \in V$ is an entity, such as a pixel or superpixel in image labeling, a line in a line drawing interpretation, or an object in scene understanding. Its variable $x_i \in \{1, 2, 3, \dots, K_i\}$ is a label or color. MRFs and CRFs in the literature belong to this category, and the task is to color the nodes V in K colors.

Candidacy graph, where each node $v_i \in V$ is a candidate or hypothesis, such as a potential label assignment for an entity, an object instance detected by bottom-up methods, or a potential match of a point to another point in graph matching. Its variable $x_i \in \{'on', 'off'\}$ is a boolean which confirms ('on') or rejects ('off') the candidate. In other words, the graph is labeled with $K = 2$

colors. In the graph matching literature [38], the candidacy graph is represented by a assignment matrix.

An adjacency graph can always be transferred to a bigger candidacy graph by converting each node v_i into K_i nodes $\{x_{ij}\}$. $x_{ij} \in \{'on', 'off'\}$ represents $x_i = j$ in the adjacency graph. These nodes observe a mutual exclusion constraint to prevent fuzzy assignments to x_i .

Fig. 6.27 shows this conversion. The adjacency graph $G_{\text{adj}} = \langle V_{\text{adj}}, E_{\text{adj}} \rangle$ has six nodes $V_{\text{adj}} = \{A, B, C, D, E, F\}$ and each has 3 ~ 5 potential labels. The variables are $X_{\text{adj}} = (x_A, \dots, x_F)$ with $x_A \in \{1, 2, 3, 4, 5\}$ and so on. We convert it to a candidacy graph $G_{\text{can}} = \langle V_{\text{can}}, E_{\text{can}} \rangle$ with 24 nodes $V_{\text{can}} = \{A_1, \dots, A_5, \dots, F_1, \dots, F_4\}$. Node A_1 represents a candidate hypothesis that assigns $x_A = 1$. The $X_{\text{can}} = (x_{A_1}, \dots, x_{F_4})$ are boolean variables.

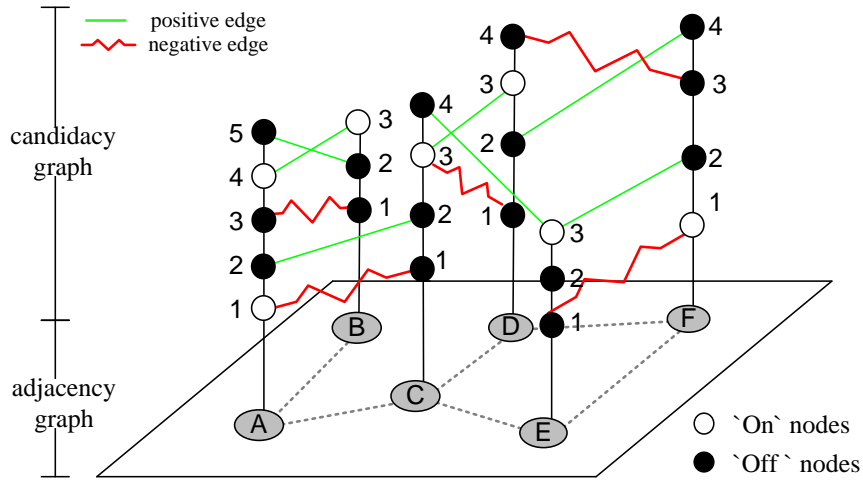


Figure 6.27: Converting an adjacency graph to a candidacy graph. The candidacy graph has positive (straight green lines) and negative (wiggled red lines) edges depending on the values assigned to the nodes in the adjacency graph.

Represented by the graph G , the vision task is posed as an optimization problem that computes a most probable interpretation with a posterior probability $p(X|\mathbf{I})$ or an energy function $\mathcal{E}(\mathcal{X})$.

$$X^* = \arg \max p(X|\mathbf{I}) = \arg \min \mathcal{E}(X). \quad (6.78)$$

To preserve the ambiguity and uncertainty, we may compute multiple distinct solutions $\{X_i\}$ with weights $\{\omega_i\}$ to represent the posterior probability.

$$(X_i, \omega_i) \sim p(X|\mathbf{I}), \quad i = 1, 2, \dots, K. \quad (6.79)$$

Positive and Negative Edges

In conventional vision formulation, edges in the graphs are a representational concept and the energy terms in \mathcal{E} are defined on the edges to express the interactions between nodes. In contrast, Swendsen-Wang [180] and Edward-Sokal [59] added a new computational role to the edges in their cluster sampling method. The edges are turned 'on' and 'off' probabilistically to dynamically form groups (or clusters) of nodes which are strongly coupled. We will introduce the clustering procedure shortly after the example below. In this section, we adopt this notion and the edges in graph G are characterized in three aspects:

Positive vs negative. A positive edge represents a cooperative constraint for two nodes having the same label in an adjacency graph or being turned on (or off) simultaneously in a candidacy graph. A negative edge requires the two nodes to have different labels in an adjacency graph or requires one node to be turned on and the other turned off in a candidacy graph.

Hard vs soft. Some edges represent hard constraints which must be satisfied, for example, in line drawing interpretation or scene labeling, while other edge constraints are soft and can be expressed with a probability.

Position dependent vs value dependent. Edges in adjacency graphs are generally *position dependent*. For example, in an Ising model an edge between two adjacent nodes poses a soft constraint that they should have the same label (ferromagnetism) or opposite labels (antiferromagnetism). In contrast, edges in candidacy graphs are *value dependent* and thus have more expressive power. This is common for vision tasks, such as scene labeling, line drawing interpretation, and graph matching. As Fig. 6.27 illustrates, the edges between nodes in the candidacy graph could be either positive or negative depending on the values assigned to nodes A, B in the adjacency graph.

As we will show in a later subsection that the positive and negative edges are crucial for generating connected components and resolving the problem of node coupling.

The Necker Cube Example

Fig. 6.28 shows the construction of a candidacy graph G for interpreting the Necker cube. For clarity of discussion we assume the exterior lines are labeled and the task is to assign two labels (concave and convex) to the six inner lines such that all local and global constraints are satisfied. Therefore we have a total of 12 candidate assignments or nodes in G .

Based on the theory of line drawing interpretation [130,178], the two 'Y'-junctions pose positive constraints so that lines 1-2-3 have the same label and lines 4-5-6 have the same label. We have 12 positive edges (green) in G to express these constraints. The intersection of lines 2 and 4 poses negative constraints that lines 2 and 4 have opposite labels which are shown in the red and wiggly edges in Fig. 6.28. The same is true for lines 3 and 6. The two different assignments for each line should also be linked by a negative edge. These negative edges are not shown for clarity.

In this candidacy graph, the two solutions that satisfy all constraints are represented by the 2-colors in Fig. 6.28. The first has all nodes 1,2, and 3 labeled convex ('x') and all nodes 4,5, and 6 labeled concave ('o'). This solution is currently in the 'on' state. This would create a valid 3D interpretation where the cube is "coming out" of the page. The alternative solution has the opposite labeling, and creates a 3D interpretation of the cube "going in" to the page.

To switch from one solution to the other, we must swap the junction labels. Each set of nodes, 1-2-3 and 4-5-6, constitutes a corner of the Necker Cube and all have positive constraints between them. This indicates that we should update all of these values simultaneously. We create two *connected component* ccp_1 and ccp_2 , comprised of the coupled nodes 1-2-3 and nodes 4-5-6 respectively. If we were simply to invert the labels of ccp_1 or ccp_2 alone we would create an inconsistent interpretation where all edges in the whole graph would now have the same label. What we need to do is simultaneously swap ccp_1 and ccp_2 .

Notice that we have negative edges between nodes 2 and 4 and between nodes 3 and 6. Negative edges can be thought of as indicators of multiple competing solutions, as they necessarily dictate that groups on either end of the edge can either be ('on', 'off') or ('off', 'on'), creating two possible outcomes. This negative edge connects nodes in ccp_1 and ccp_2 , thus indicating that those nodes in the two ccp 's must have different labels. We construct a composite connected component (called $cccp$), $cccp_{12}$, encompassing nodes 1-6, we now have a full component that contains all relevant

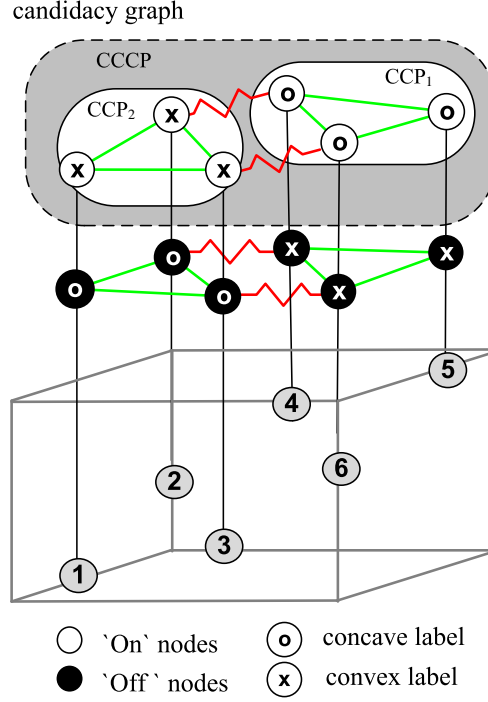


Figure 6.28: The Necker cube example. The adjacency graph with 6 nodes (bottom) is converted to a candidacy graph of 12 nodes (top) for concave and convex label assignments respectively. 12 positive and 2 negative edges are placed between these candidate assignments to ensure consistency.

constraints. Moving from solution 1 to 2 is now as simple as flipping all the nodes simultaneously, or equivalently satisfying all of the constraints.

In the next subsection, we explain how we form the *ccp*'s and *cccp*'s in a formal way.

Edge Probability for Clustering

On each positive or negative edge, we define an *edge probability* (using bottom-up discriminative models) for the coupling strength. That is, at each edge $e \in E$, we define an auxiliary probability $u_e \in \{0, 1\}$ or $\{'on', 'off'\}$, which follows an independent probability q_e .

In Swendsen and Wang [180], the definition of q_e is decided by the energy term in the Potts model $q_e = e^{-2\beta}$ as a constant for all e . barbu2005generalizing and Zhu [9], for the first time, separate q_e from the energy function and define it as a bottom-up probability: $q_e = p(l(x_i) = l(x_j)|F(x_i), F(x_j)) = p(e = on|F(x_i), F(x_j))$ with $F(x_i)$ and $F(x_j)$ being local features extracted at node x_i and x_j . This can be learned through discriminative training, for example, by logistic regression and boosting,

$$\frac{p(l(x_i) = l(x_j)|F(x_i), F(x_j))}{p(l(x_i) \neq l(x_j)|F(x_i), F(x_j))} = \sum_n \lambda_n h_n(F(x_i), F(x_j)).$$

On a positive edge $e = (i, j) \in E^+$, $u_e = 'on'$ follows a Bernoulli probability,

$$u_e \sim \text{Bern}(q_e \cdot 1(x_i = x_j)).$$

$1()$ is boolean function. It equals 1 if the condition is satisfied and 0 otherwise. Therefore, at the present state X , if the two nodes have the same color, i.e. $x_i = x_j$, then the edge e is turned on with probability q_e . If $x_i \neq x_j$, then $u_e \sim \text{Bern}(0)$ and e is turned off with probability 1. So, if two nodes are strongly coupled, q_e should have a higher value to ensure that they have a higher probability to stay the same color.

Similarly, for negative edges $e \in E^-$, $u_e = \text{'on'}$ also follows a Bernoulli probability,

$$u_e \sim \text{Bern}(q_e 1(x_i \neq x_j)).$$

At the present state X , if the two nodes have the same color $x_i = x_j$, then the edge e is turned off with probability 1, otherwise e is turned on with probability q_e to enforce that x_i and x_j stay in different colors.

After sampling u_e for all $e \in E$ independently, we denote the sets of positive and negative edges that remain 'on' as $E_{on}^+ \subset E_+$ and $E_{on}^- \subset E^-$ respectively. Then we have a formal definitions of the *ccp* and *cccp*.

Definition 6.2. A *ccp* is a set of vertices $\{v_i; i = 1, 2, \dots, k\}$ for which every vertex is reachable from every other vertex by the positive edges in E_{on}^+ .

Definition 6.3. A *cccp* is a set of *ccp*'s $\{ccp_i; i = 1, 2, \dots, m\}$ for which every *ccp* is reachable from every other *ccp* by the negative edges in E_{on}^- .

No two *ccp*'s are reachable by positive edges, or else they would be a single *ccp*. Thus a *cccp* is a set of isolated *ccp*'s that are connected by negative edges. An isolated *ccp* is also treated as a *cccp*.

In section 6.9.6, we will treat the invalid cases where a *ccp* contains negative edges by converting it to a *cccp*.

To observe the detailed balance equations in MCMC design, we need to calculate the probabilities for selecting a *ccp* or *cccp* which are determined by the edge probabilities q_e . For this purpose we define their cuts. In general, a cut is the set of all edges connecting nodes between two nodes sets.

Definition 6.4. Under a current state X , a *cut* for a *ccp* is the set all positive edges between nodes in *ccp* and its surrounding nodes which have the same label,

$$Cut(ccp|X) = \{e : e \in E^+, x_i = x_j, i \in ccp, j \notin ccp\}.$$

These are the edges that must be turned off probabilistically (with probability $1 - q_e$) in order to form the *ccp* and the cut depends on the state X .

Definition 6.5. A cut for a *cccp* at a state X is the set of all negative (or positive) edges connecting the nodes in the *cccp* and its neighboring node which have different (or same) labels,

$$\begin{aligned} Cut(cccp|X) = & \{e : e \in E^-, i \in cccp, j \notin cccp, x_i \neq x_j\} \\ & \cup \{e : e \in E^+, i \in cccp, j \notin cccp, x_i = x_j\}. \end{aligned}$$

All these edges must be turned off probabilistically with probability $1 - q_e$ in order to form the composite connected component *cccp* at state X .

As edges in E_{on}^+ only connect nodes with the same label, so all nodes in a *ccp* have the same label. In contrast, all edges in E_{on}^- only connect nodes with different labels, adjacent *ccp*'s in a *cccp* must have different labels.

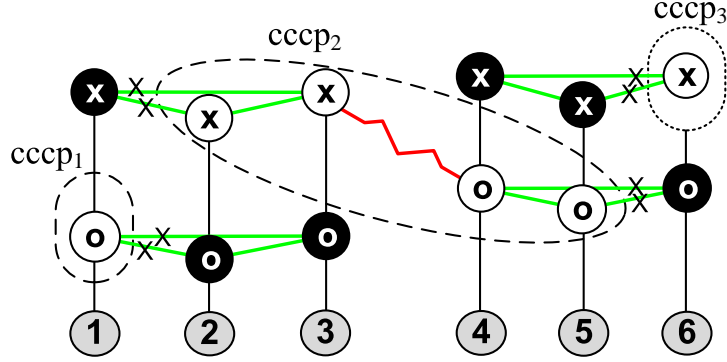


Figure 6.29: A Necker cube candidacy graph not in a solution state.

To illustrate the concepts, we show a non-solution state X for the Necker cube in Figure 6.29. By turning off some edges (marked with the crosses), we obtain three $cccp$'s for the nodes that are currently 'on'. In this example, $q_e = 1$, as these are hard constraints that are inviolable. $cccp_1$ and $cccp_3$ have only 1 node, and $cccp_2$ has two ccp 's with 4 nodes. The algorithm will now arbitrarily select a $cccp$ and update its values according to its constraints. If it selects either $cccp_1$ or $cccp_3$, then we are one step closer to the solution. If it selects ($cccp_2$), then all the 4 vertex labels are swapped and we have reached a solution state and will continue to swap back and forth between the two solutions.

6.9.3 C^4 algorithm on flat graphs

In this section, we introduce the C^4 algorithm for cluster sampling on flat graphs.

Outline of the algorithm

The C^4 algorithm works iteratively following the MCMC design. In each iteration, it generates the $cccp$'s, selects (or visits) a $cccp_o$ with a probability, and reassigns labels to its ccp 's such that all internal negative constraints are satisfied. As the number of ccp 's in $cccp_o$ grows large, the number of potential labelings will grow as well. One can remedy this situation in two ways:

1. Use a constraint-satisfaction problem (CSP)-solver to solve this smaller, easier constraint satisfaction problem within $cccp_o$.
2. Use random or heuristic sampling to find a new valid labeling.

We will use the second approach throughout this section and the number of ccp 's in a $cccp_o$ is in general small, so the label assignment is not a problem. The C^4 algorithm can be viewed as a method that breaks a large constraint-satisfaction problem into smaller fragments in $cccp_o$ which can be satisfied locally. Then it propagates the solution through iterations.

This assignment represents a move in MCMC which is accepted by the Metropolis-Hastings step with an acceptance probability. The acceptance probability account for the probabilities for generating the $cccp$'s, selecting a $cccp_o$, assigning new labels, and the posterior probability.

In summary, we state the C^4 algorithm below.

We will elaborate on the probabilities used in the algorithm in the next subsection,

Input: A graph $G = \langle V, E \rangle$ and posterior prob. $p(X|\mathbf{I})$.

Calculate the edge probability $q_e, \forall e \in E$.

q_e is a problem specific discriminative probability.

Initialize the state $X = (x_1, x_2, \dots, x_n)$.

e.g. all nodes are turned off in a candidacy graph.

for $s = 1$ to N^{iter} **do**

Denote the current X by state A .

Step 1: Generate a $cccp_o$ at state A

$\forall e = (i, j) \in E^+$, sample $u_e \sim \text{Bern}(q_e 1(x_i = x_j))$

$\forall e = (i, j) \in E^-$, sample $u_e \sim \text{Bern}(q_e 1(x_i \neq x_j))$

Generate the $\{ccp\}$ and $\{cccp\}$ based on E_{on}^+ and E_{on}^-

Select a $cccp_o$ from $\{cccp\}$ probabilistically

Denote the prob for selecting $cccp_o$ by $q(cccp_o|A)$.

Step 2: Assign labels to ccp 's in the $cccp$ with probability: $q(l(cccp_o = L|cccp_o, A))$.

Denote the new X as state B .

Step 3: Calculate the acceptance probability:

$$\alpha(A \rightarrow B) = \min(1, \frac{q(B \rightarrow A)}{q(A \rightarrow B)} \cdot \frac{p(X = B|\mathbf{I})}{p(X = A|\mathbf{I})})$$

end for

Output: Distinct states $\{X^*\}$ with highest probabilities.

Figure 6.30: **The C^4 Algorithm**

Calculating the Acceptance Probability

In Markov chain design, each move between two states A and B is made reversible and observes the detailed balance equation,

$$p(X = A|\mathbf{I})K(A \rightarrow B) = p(X = B|\mathbf{I})K(B \rightarrow A). \quad (6.80)$$

$K(A \rightarrow B)$ is the Markov chain kernel or transition probability from A to B . In the Metropolis-Hastings design,

$$K(A \rightarrow B) = q(A \rightarrow B)\alpha(A \rightarrow B), \forall A \neq B. \quad (6.81)$$

$q(A \rightarrow B)$ is the probability for proposing state B from state A , and $\alpha(A \rightarrow B)$ is the acceptance probability,

$$\alpha(A \rightarrow B) = \min(1, \frac{q(B \rightarrow A)}{q(A \rightarrow B)} \cdot \frac{p(X = B|\mathbf{I})}{p(X = A|\mathbf{I})}). \quad (6.82)$$

It is easy to check that the design of proposal probability in eqn.(6.82) and the acceptance probability in eqn.(6.81) makes the kernel satisfy the detailed balance equation in (6.80), which in turn suffices to observe the invariance condition,

$$p(X = A|\mathbf{I})K(A \rightarrow B) = p(X = B|\mathbf{I}). \quad (6.83)$$

So, $p(X|\mathbf{I})$ is the invariant probability of the Markov chain with kernel K . Now we elaborate on the design of proposal and acceptance probabilities. The acceptance probability is determined by two ratios.

(i) The ratio $\frac{p(X=B|\mathbf{I})}{p(X=A|\mathbf{I})}$ is problem specific and is not part of our design. The posterior probability can be in general form and does not have to be modified or approximated to fit the C^4 algorithm. As states A and B only differ in their labels for nodes in $cccp_o$, it often can be computed locally if the posterior probability is a MRF or CRF.

(ii) The proposal probability ratio is completely up to our design, and it includes two parts,

$$\frac{q(B \rightarrow A)}{q(A \rightarrow B)} = \frac{q(cccp_o|B)}{q(cccp_o|A)} \cdot \frac{q(l(cccp_o) = L_A|cccp_o, B)}{q(l(cccp_o) = L_B|cccp_o, A)}.$$

$q(cccp_o|A)$ and $q(cccp_o|B)$ are the probabilities for choosing $cccp_o$ at states A and B respectively. Given the chosen composite connected component $cccp_o$, in both states A and B , the assignment of new labels is independent of the surrounding neighbors of $cccp_o$ and is often assigned by equal probability (uniform) among all valid assignments in the CSP-solver. Thus they cancel out, and we have $\frac{q(l(cccp_o)=L_A|cccp_o, B)}{q(l(cccp_o)=L_B|cccp_o, A)} = 1$.

To summarize, the key to the algorithm design is the ratio $\frac{q(cccp_o|B)}{q(cccp_o|A)}$. In single site sampling, such as Gibbs sampler, each node is a $cccp_o$ and the selection is simply a visiting scheme. In C^4 , the probability for choosing $cccp_o$ at a state depends on two steps: (a) How likely it is to generate $cccp_o$ by sampling the edge probabilities q_e following the Bernoulli probability. (b) How likely it is to select $cccp_o$ from the set of formed $\{cccp\}$ in states A and B . These probabilities are hard to compute, because there are a vast amount of partitions of the graph that include a certain $cccp_o$ by turning on/off edges. A partition is a set of $cccp$'s after turning off some edges.

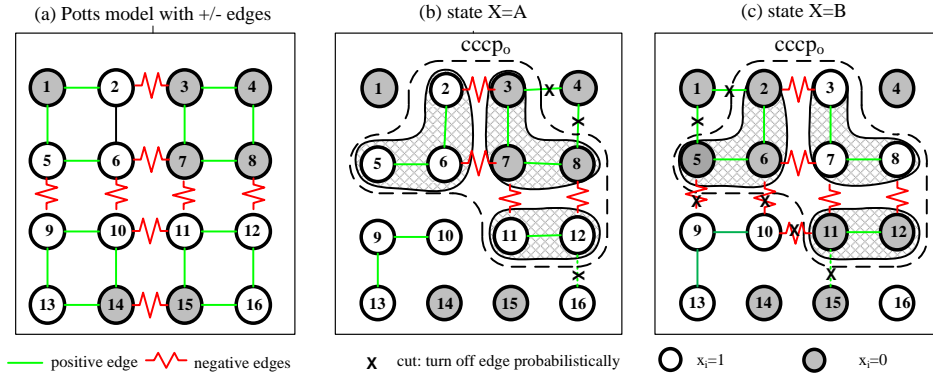


Figure 6.31: The Potts model with negative edges. (a) Minimum energy is a checkerboard pattern. (b) Forming $cccp$ s. (c) $cccp_o$ consists of sub- $cccp$ s of positive edges connected by negative edges.

Interestingly, the set of all possible partitions in state A is identical to those in state B , and all these partitions must share the same cut $Cut(cccp_o)$. That is, in order for $cccp_o$ to be a composite connected component, its connections with its neighboring nodes must be turned off. Even though the probabilities are in complex form, their ratio is simple and clean due to cancellation. Furthermore, given the partition, $cccp_o$ is selected with uniform probability from all possible $cccp$'s.

Proposition 1. The proposal probability ratio for selecting $cccp_o$ at states A and B is

$$\frac{q(cccp_o|B)}{q(cccp_o|A)} = \frac{\prod_{e \in Cut(cccp_o|B)} (1 - q_e)}{\prod_{e \in Cut(cccp_o|A)} (1 - q_e)}. \quad (6.84)$$

We will prove this in the appendix in a similar way to the SW-cut method [9]

Special case: Potts model with +/- edges

To illustrate C^4 , we derive it in more detail for a Potts model with positive and negative edges. Let X be a random field defined on a 2D lattice with discrete states $x_i \in \{0, 1, 2, \dots, L-1\}$. Its probability is specified by

$$p(X) = \frac{1}{Z} \exp\{-\mathcal{E}(X)\}; \quad (6.85)$$

$$\mathcal{E}(X) = \sum_{\langle i,j \rangle \in E^+} \beta \delta(x_i = x_j) + \sum_{\langle i,j \rangle \in E^-} \beta \delta(x_i \neq x_j),$$

where $\beta > 0$ is a constant. The edge probability will be $q_e = 1 - e^{-\beta}$ for all edges.

Fig. 6.31.(a) shows an example on a small lattice with $L = 2$ labels, which is an adjacency graph with position dependent edges. The states with checkerboard patterns will have highest probabilities. Fig. 6.31(b) and (c) show two reversible states A and B by flipping the label of a $cccp_o$ in one step. In this example, $cccp_o$ has three ccp 's, $cccp_o = \{\{2, 5, 6\}; \{3, 7, 8\}; \{11, 12\}\}$. The labels of the 8 nodes are re-assigned with uniform probability, and this leads to the difference in the cuts for $cccp_o$ at the two states, $Cut(cccp_o|A) = \{(3, 4), (4, 8), (12, 16)\}$ and $Cut(cccp_o|B) = \{(1, 2), (1, 5), (5, 9), (6, 10), (10, 11), (11, 15)\}$.

Proposition 2. The acceptance probability for C^4 on the Potts model is $\alpha(A \rightarrow B) = 1$ for any two states with different labels in $cccp_o$. Therefore, the move is always accepted.

The proof follows two observations. Firstly, the energy terms inside and outside $cccp_o$ are the same for both A and B , and they differ only at the cuts of $cccp_o$. More precisely, let $c = |Cut(cccp_o|B)| - |Cut(cccp_o|A)|$ be the difference of sizes in the two cuts (i.e. $c = 3$ in our example), it is not too hard to show that

$$\frac{p(X = B|\mathbf{I})}{p(X = A|\mathbf{I})} = e^{-\beta c} \quad (6.86)$$

Secondly, we have the proposal probability ratio, following eqn.(6.84),

$$\frac{q(cccp_o|B)}{q(cccp_o|A)} = \frac{(1 - q_e)^{|Cut(cccp_o|B)|}}{(1 - q_e)^{|Cut(cccp_o|A)|}} = e^{\beta c}. \quad (6.87)$$

Plugging the two ratios in eqn.6.82, we have $\alpha(A \rightarrow B) = 1$. In the literature of SW [59], Edwards and Sokal explain the SW on Potts model as data augmentation where the edge variables $\{u_e\}$ are treated as auxiliary variables and they sample $\{x_i\}$ and $\{u_e\}$ iteratively from a joint probability.

6.9.4 Experiments on Flat Graphs

In this section we test C^4 's performance on some flat graphs (MRF and CRF) in comparison with the Gibbs sampler [75], SW method [180], iterated conditional modes (ICM), graph cuts [23], and loopy belief propagation (LBP) [108]. We choose classical examples: (i) Ising/Potts model for MRF; (ii) Line drawing interpretation for constrained-satisfaction problem using candidacy graph; (iii) scene labeling using CRF; and (iv) scene interpretation of aerial images.

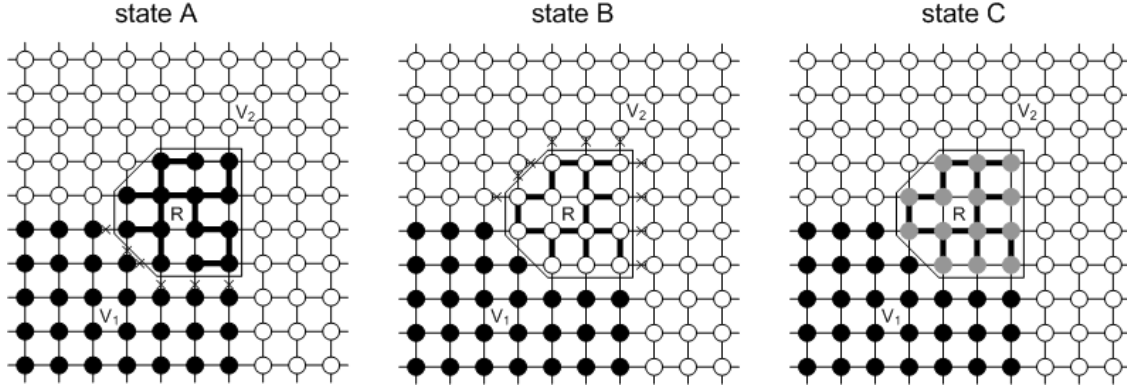


Figure 6.32: The Ising/Potts model with checkerboard constraints and two minimum energy states computed by C^4 .

6.9.5 Checkerboard Ising Model

We first show the Ising model on a 9×9 lattice with positive and negative edges (the Ising model is a special case of the Potts model with $L = 2$ labels). We tested C^4 with two parameters settings: (i) $\beta = 1$ and thus $q_e = 0.632$; and (ii) $\beta = 5$ and thus $q_e = 0.993$. In this lattice we've created a checkerboard pattern. We've assigned negative and positive edges so that blocks of nodes want to be the same color, but these blocks want to be different colors than their neighbors.

Fig. 6.32 shows a typical initial state to start the algorithm, and two solutions with minimum (i.e. 0) energy. Fig. 6.33(a) shows a plot of energy versus time for C^4 , Gibbs sampler, SW, graph cuts, and LBP. C^4 converges second fastest of all five algorithms in about 10 iterations, behind graph cuts. Belief propagation cannot converge due to the loopiness of the graph, and Gibbs sampler and the conventional Swendsen-Wang cannot quickly satisfy the constraints as they do not update enough of the space at each iteration. This shows that C^4 has a very low burn-in time.

Fig. 6.33(b)(c) show the state visited at each iteration. We show the states in 3 levels: the curve hits the ceiling or floor for the two minimum energy states respectively, and the middle for all other states. Here we are only comparing graph cuts, SW and C^4 as they are the only algorithms that converge to a solution in a reasonable amount of time. C^4 keeps swapping solutions while SW and graph cuts get stuck in their first solution. This is because C^4 can group along negative edges as well as positive edges to update large portions of the system at once, while Swendsen-Wang is stuck proposing low probability moves over smaller portions of the solution space.

We also compared our results for experiments where $\beta = 1$ and $\beta = 5$. Figure 6.33(c) shows the states visited by the sampler over time. In the $\beta = 1$ case, it takes longer for C^4 to converge, because it can't form large components with high probability. As β gets large, however, C^4 very quickly takes steps in the space towards the solution and can move rapidly between solution states. We have found that an annealing schedule where $q_e = 1 - e^{-\beta/T}$ and T is adjusted such that q_e moves from 0 to 1 over the course of the experiment works quite well too.

We finally compare the estimated marginal beliefs at each node as computed by each algorithm. LBP computes these beliefs directly, but we can estimate them for Gibbs sampling, SW, and C^4 by running each algorithm and recording the empirical mean at each iteration for each node given the previous states. Fig. 6.34 shows the belief for one of the Ising model sites over time for each of the 4 algorithms. LBP does not converge, so it has a noisy estimate over time and is not plotted for clarity, Gibbs and SW converge to a probability of 1, because they get stuck in a single solution

state, while C^4 approaches 0.5, as it keeps flipping between the two states.

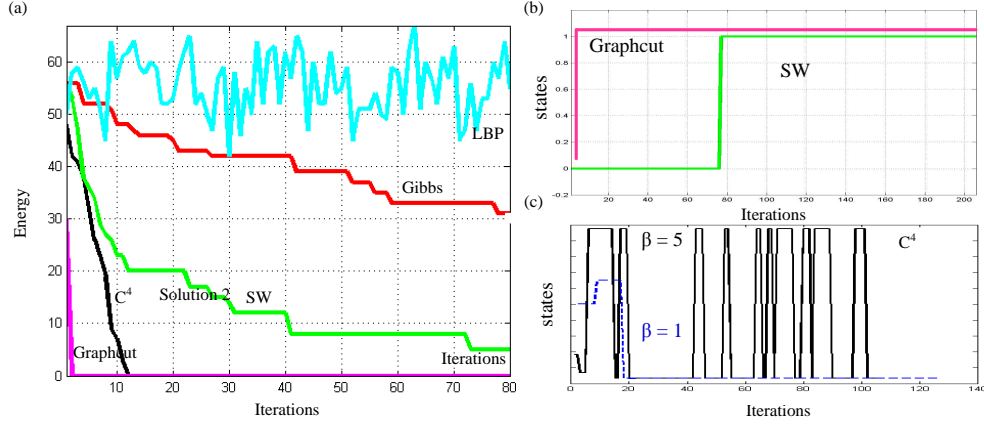


Figure 6.33: (a) Energy plots of C^4 , SW, Gibbs sampler, graph cuts, and LBP on the Ising model vs. time. (b) (c) The state (visited by the algorithms) in time for graph cuts, SW and C^4 . Once SW and graph cuts hit the first solution, they get stuck while C^4 keeps swapping between the two minimum energy states. C^4 results shown for $\beta = 1$ and $\beta = 5$.

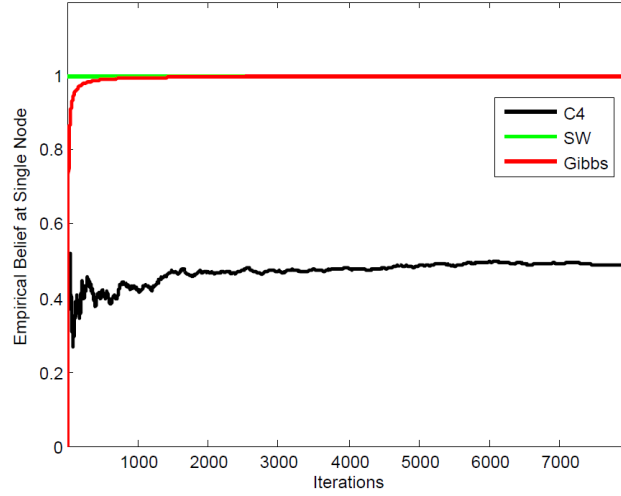


Figure 6.34: Comparison of marginal beliefs at a single site of the Ising model for Gibbs, SW, and C^4 . C^4 correctly converges toward 0.5, while the other algorithms only find a single solution state. LBP does not converge and thus has erratic beliefs that we do not show on this plot.

Checkerboard Potts Model with 7 Labels

We ran the same experiment as with the Ising model above but this time solved the same checkerboard pattern on a Potts model in which each site could take one of seven possible colors ($L = 7$). In this example, we have a large number of equal states (in checkerboard pattern) with minimum energy.

Fig. 6.35(a) plots the energy convergence of each algorithm over time. Graph cuts again converges to just one of the many solutions. Unlike in the case of the $L = 2$ model, SW is able to find

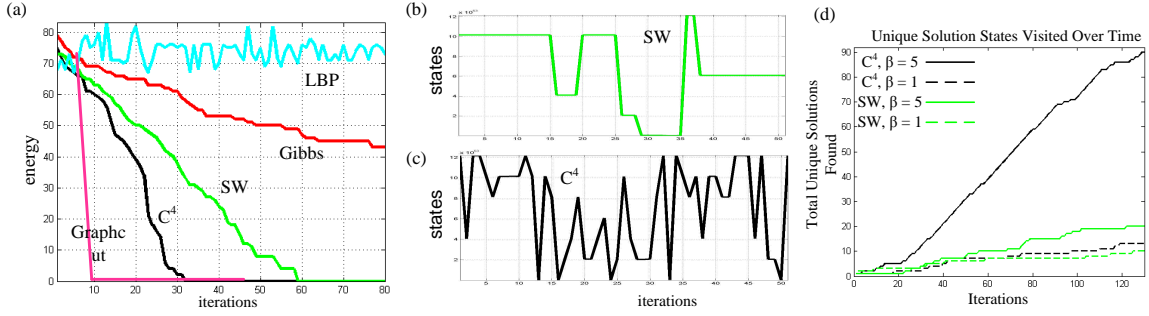


Figure 6.35: (a) Energy plots of C^4 , SW, Gibbs sampler, and LBP on the Potts model ($L = 7$) vs. time. (b) (c) The minimum energy states visited by SW and C^4 algorithms over time. (d) Total number of unique solutions found vs. time for SW and C^4 with $\beta = 1$ and $\beta = 5$.

multiple solutions this time, as seen in Fig. 6.35(b). Fig. 6.35(c) shows the number of distinct states with minimum energy that have been visited by SW and C^4 over time. We see that C^4 explores more states in a given time limit which again demonstrates that C^4 is more dynamic and thus have fast mixing time – a crucial measure for the efficiency of MCMC algorithms. We also compare the case where $\beta = 1$ vs. $\beta = 5$. Once again, we see that $\beta = 1$ doesn't create strong enough connections for C^4 to move out of local minimum, so it finds roughly as many unique solutions as Swendsen-Wang does (about 13). When β is increased to 5, however, the number skyrockets from 13 to 90. We thus see that C^4 can move around the solution space much more rapidly than other methods when β is high and can discover a huge number of unique solution states.

Line Drawing Interpretation

The previous two examples are based on MRF models whose edges are position dependent. Now we test on line drawing interpretation on candidacy graph. We use two classical examples which have multiple stable interpretations, or solutions: (i) the Necker cube in Fig. 6.25 that has two interpretations; and (ii) a line drawing with double cubes in Fig. 6.36 that has four interpretations. The swapping between these states involves the flipping of 3 or 12 lines simultaneously. Our goal is to test whether the algorithms can compute the multiple distinct solutions over time.

We adopt a Potts like model on the candidacy graph. Each line in the line drawing is a node in the Potts model, which can take one of eight line drawing labels indicating whether the edge is concave, convex, or a depth boundary. See [178] for an in-depth discussion on labels for consistent line drawings. We add an edge in our candidacy graph between any two lines that share a junction. At each junction, there are only a small set of valid labels for each line that are realizable in a 3D world. We add positive edges between pairs of line labels that are consistent with one of these junction types, and negative edges between line labels that are not. Thus, we model the pairwise compatibility of neighboring line labels given the type of junction they form.

For these experiments we set $\beta = 2$, resulting in $q_e = 0.865$. Figs 6.36(a) and (c) plot the state visited by the algorithms over time. Once again we see that C^4 can quickly switch between solutions where CSP solvers or other MCMC methods could get stuck.

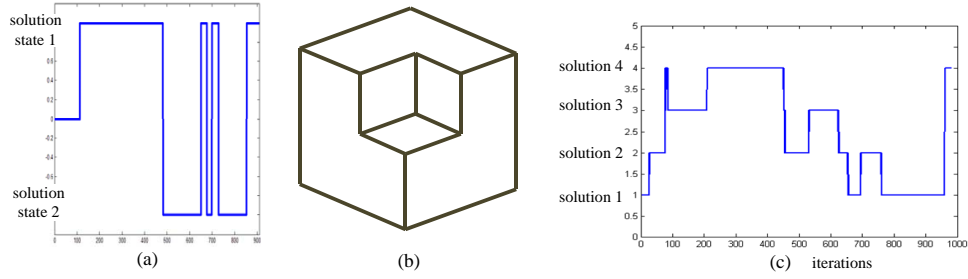


Figure 6.36: Experimental results for swapping state between interpretations: (a) States visited by C^4 for the Necker Cube. (b) A line drawing with outer and inner cubes. (c) States visited by C^4 for the double cubes.

Parsing Aerial Images

In this experiment we use C^4 to parse aerial images. This experiment is an extension of our work from [162]. In [162], aerial images are represented as collections of groups of objects, related by statistical appearance constraints. These constraints are learned automatically in an offline phase prior to inference.

We create our candidacy graph by letting each bottom-up detected window be a vertex in the graph, connected by edges with probabilities proportional to how compatible those objects are (we refer to [162] for detailed discussion of the energy function). Each candidate can be on or off, indicating whether it is in the current explanation of the scene or not.

Each edge is assigned to be positive or negative and assigned a probability q_e of being on by examining the energy $e = \phi(x_i, x_j)$ between its two nodes. If $e > t$, the edge is labeled as a negative edge and if $e < t$ the edge is labeled as a positive edge, where t is a threshold of the user's choosing. In our experiments we let $t = 0$. In this way we create data-driven edge probabilities and determine positive and negative edge types for C^4 .

In these experiments we learned a prior model for likely object configurations using labeled aerial images. Object boundaries were labeled in each image from a set of over 50 images. We tested the results on five large aerial images collected from Google Earth that were also labeled by hand, so that we could measure how much C^4 improved the final detection results. Though we only use five images, each image is larger than 1000×1000 pixels and includes hundreds of objects, so one could also think of the evaluation as spanning 125 images of 200×200 pixels.

Figure 6.37 shows an example of a parsed aerial scene. The bottom-up detected windows are treated as candidates and many are false positives. After using C^4 minimizing a global energy function, however, we are left with the subset that best satisfies the constraints of the system. The false positive rates are vastly diminished after C^4 rules out incompatible proposals. Figure 6.37(d) shows the precision-recall curve for aerial image object detection using C^4 vs. just bottom-up cues. We can see that the C^4 curve, drawn in dashed green, has a much higher precision than the bottom-up detections even as the recall increases.

We also compared the results of using C^4 over LBP, ICM, and SW for similar false alarm rates. The results are shown in Table 6.3

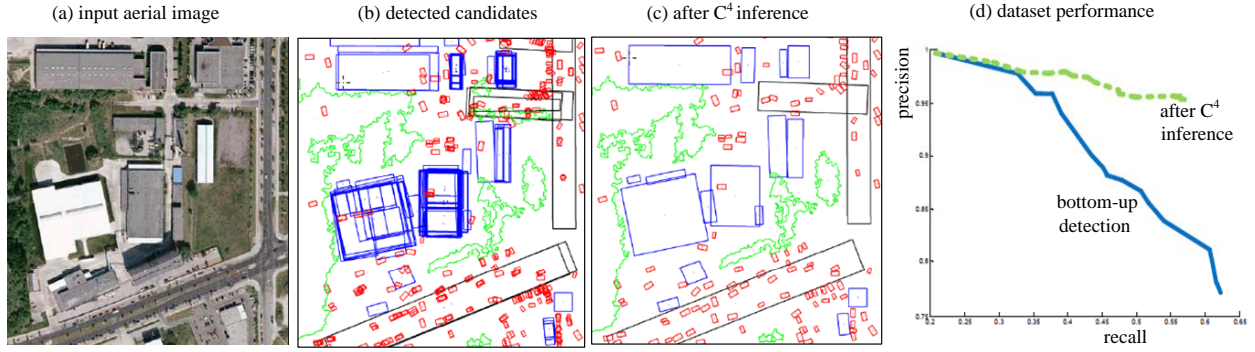


Figure 6.37: An application of C^4 to aerial image parsing. (a) A portion of an aerial image from Google earth. (b) A set of bottom-up detections of objects with each being a candidate, i.e. a node in the candidacy graph. Note the large number of false positives that need to be turned off. (b) The final subset of proposals selected by C^4 to represent the scene. C^4 has removed the candidates that are inconsistent with the prior. (c) Precision recall curve for pixel level performance over a dataset of aerial images.

Method	FalsePositive (per image)	DetectRate (%)
LBP	85.32	0.668
ICM	82.11	0.768
SW	87.91	0.813
C^4	83.04	0.875

Table 6.3: False positives per image and detection rate using Loopy BP, SW, ICM, and C^4 for aerial image parsing.

6.9.6 C^4 on Hierarchical Graphs

In this section, we discuss the consistency of the flat graphs and extend C^4 from flat graphs to hierarchical graphs and then we address high-order constraints that involve more than two sites.

Condition for Graph Consistency

In each iteration of the C^4 algorithm, suppose we have turned on edges probabilistically and the original graph $G = \langle V, E \rangle$ becomes $G_{on} = \langle V, E_{on} \rangle$ with $E = E_{on} \cup E_{off}$, $E_{on} = E_{on}^+ \cup E_{on}^-$, and $E_{off} = E_{off}^+ \cup E_{off}^-$. As we discussed in section 6.9.2 all nodes in the graph G_{on} in each ccp shares the same label and they are supposed to form a coupled partial solution. However, if the constraints in graph G are inconsistent, then some nodes in a ccp may be connected by edges in E_{off}^- . Though such negative edges are not turned on in ccp , they indicate that some nodes in the ccp may be conflicting to each other. This may not be a serious problem, for example, the negative edges may simply express soft constraints, such as overlapping windows due to occlusion, which is acceptable in the final solution.

Fig. 6.38 shows an example where the negative edge is a hard constraint. If we try to solve the duck/rabbit illusion using flat candidacy graph, a ccp may contain $\{ 'eye', 'nose', 'head' \}$ which is inconsistent. We call it a "love triangle".

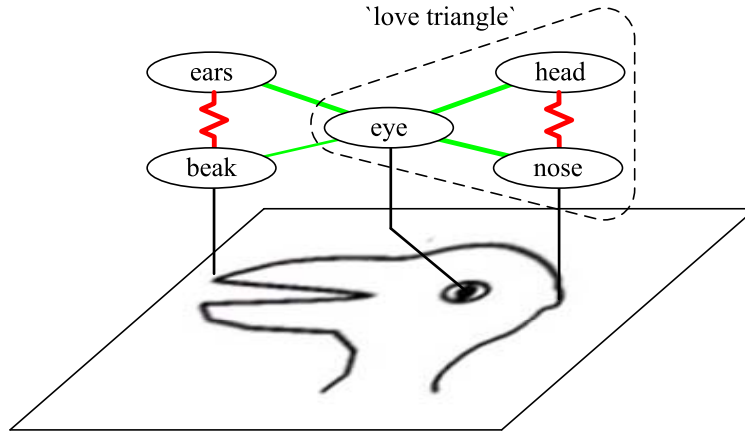


Figure 6.38: An attempt to solve the duck/rabbit illusion using flat C^4 . We see that we are very likely to form love triangles on the left and right of the graph, making constraint satisfaction very difficult.

Definition 6.6. In a graph G , two nodes i, j connected by a negative edge is said to be involved in a *love triangle* if there also exists a path between i, j that consists of all positive edges.

Definition 6.7. A *ccp* is said to be *consistent* in graph G if there are no negative edges in E that connect two nodes in the *ccp*, that is, $\{e : i, j \in ccp\} \cap E^- = \emptyset$. A graph G is said to be *consistent* if all its *ccp*'s are always consistent in C^4 .

When a graph is consistent, then we are guaranteed to get valid solutions.

The existence of the so-called 'love triangles' are the sole reason to generate inconsistent *ccp*'s. For this we can easily prove the following proposition.

Proposition 3. In the absence of 'love triangles', the graph G will be consistent.

The essential reason for generating the 'love triangles' in a graph, mostly in candidacy graphs, is that certain nodes are over-loaded with multiple labels and thus they are coupled with conflicting nodes. For example, the node 'eye' should be either a 'rabbit eye' or a 'duck eye' and it should be split into two conflicting candidates connected by an negative edge. This way it can eliminate the "love triangle". Fig. 6.39 illustrates that we can remove the love triangle by splitting node 1 into nodes 1 and 1' and thus we will have consistent *ccp*.

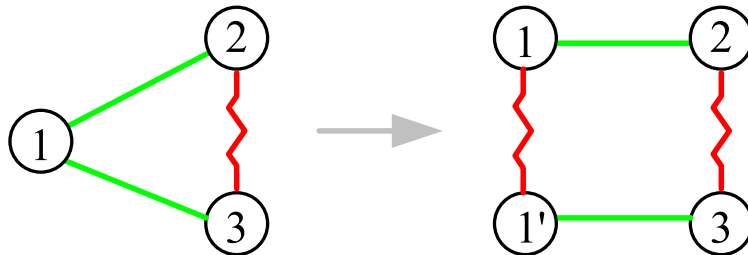


Figure 6.39: Breaking the 'love triangle' in a candidacy graph.

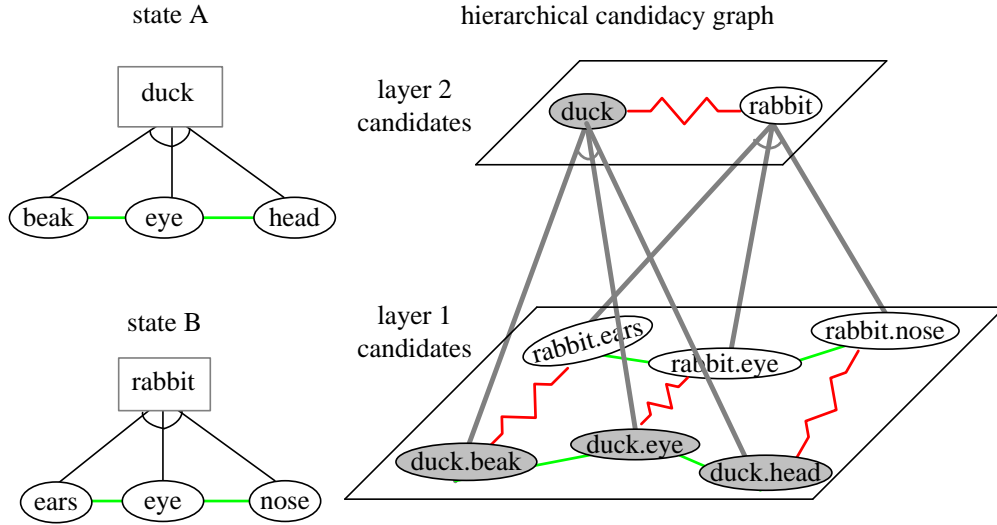


Figure 6.40: An attempt to solve the duck/rabbit illusion using hierarchical C^4 . The trees define which parts comprise each object. Nodes are grouped according to these trees, creating higher-level nodes. The higher-level nodes inherit the negative constraints.

Formulation of Hierarchical C^4

One other common issue that we need to address is higher-order constraints that involve more than 2 nodes. Fig. 6.40 shows a hierarchical graph representation for the duck/rabbit illusion. This is a candidacy graph with two layers. The top layer contains two hidden candidate hypotheses: 'duck' and 'rabbit'. The two nodes are decomposed into three parts in layer 1 respectively and thus impose high order constraints between them. Now the hypotheses for parts are specifically for 'duck.eye', 'rabbit.eye' etc. The negative edge connecting the two object nodes is inherited from their overlapping children.

This hierarchical candidacy graph is constructed on-the-fly with nodes being generated by multiple bottom-up detection and binding processes as well as top-down prediction processes. We refer to a recent paper by Wu and Zhu [207] for the various bottom-up/top-down processes in object parsing. In this graph, positive and negative edges are added between nodes on the same layers in a way identical to the flat candidacy graph, while the vertical links between parent-child nodes are deterministic.

By turning on/off the positive and negative edges probabilistically at each layer, C^4 obtains *ccp*'s and *cccp*'s as in the flat candidacy graphs. In this case, a *ccp* contains a set of nodes that are coupled in both horizontal and vertical directions and thus represents a partial parse tree. A *cccp* contains multiple competing parse trees, which will be swapped in a single step. For example, the left panel in Fig. 6.40 shows two *ccp*'s for the duck and rabbit respectively which are connected with negative edges in the candidacy graph.

This hierarchical representation can also eliminate the inconsistency caused by overloaded labels. That is, if a certain part is shared by multiple object or object instances, we need to create multiple instances as nodes in the hierarchical candidacy graph.

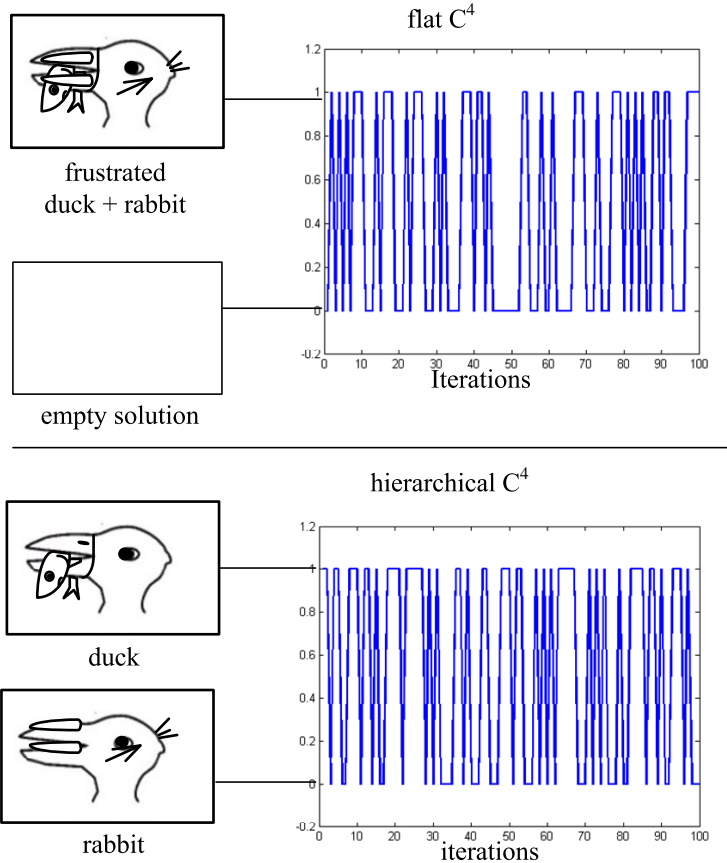


Figure 6.41: (Top panel) Flat C^4 results on the duck/rabbit illusion. C^4 swaps between two impossible states due to love triangles. (Bottom panel) Hierarchical C^4 results on the duck/rabbit solution. C^4 now swaps uniformly between the two correct solutions.

6.9.7 Experiments on Hierarchical C^4

To demonstrate the advantages of hierarchical C^4 over flat C^4 , we present two experiments (i) interpreting the duck/rabbit illusion, and (ii) finding configurations of object parts amidst extremely high noise.

(i) Experiment on Hierarchical Duck/Rabbit Illusion. As referenced above, C^4 on the flat candidacy graph in Fig. 6.38 creates two love triangles. The top panel of Fig. 6.41 shows the results of flat C^4 on the duck/rabbit illusion. C^4 continuously swaps between two states, but the two states either have all nodes on or all nodes off, neither of which are valid solutions. The bottom panel of Figure 6.41 shows the results of applying hierarchical C^4 to the duck/rabbit illusion. We defined a tree for the duck/rabbit illusion consisting of either a duck, $\{beak, eye, duck_{head}\}$, or a rabbit $\{ears, eye, rabbit_{head}\}$. As a result, the algorithm instantly finds both solutions and then proceeds to swap between them uniformly. These results show that hierarchical C^4 can help guide the algorithm to more robust solutions and negates the effects of love triangles.

(ii) Experiments on Object Parsing. A problem that often appears in computer science is the problem of finding the optimal subset from a larger set of items that minimizes some energy function. For example, in the star model [63], many instances of each object part may be detected in the image. However, our algorithm should find the subset (or subsets) of these detections that creates the highest probability configuration. This is a combinatorially hard problem as the number

of solutions grows exponentially in the number of detections, so heuristic approaches are usually proposed to deal with this situation. One can use dynamic programming for inferring star models, but these require that the root part be present, which our algorithm does not. Hierarchical C^4 is ideally suited for this problem, as it can use local edge constraints and hierarchical grouping to guide its search through large sets of detections to find the most likely solutions.

Chapter 7

Convergence Analysis of MCMC

Let (ν, K, Ω) be a Markov chain with initial distribution ν , transition kernel K in the space Ω . The samples obtained by this Markov chain at some time n follow the distribution $X(t) \sim \nu \cdot K^n \xrightarrow{n \rightarrow \infty} \pi$.

The convergence of νK^n is measured using the total variation $\|\nu K^n - \pi\|_{TV} \rightarrow 0$.

$$K^n = \sum_{i=1}^n \lambda_i v_i u_i$$

Recall the typical measurements, define in Section 3.3:

i) First hitting time of a state i (In the finite state case)

$$\tau_{\text{hit}}(i) = \inf\{n \geq 1; x_n = i, x_0 \sim \nu_0\}, \quad \forall i \in \Omega$$

ii) First return time of a state i

$$\tau_{\text{ret}}(i) = \inf\{n \geq 1; x_n = i, x_0 = i\}, \quad \forall i \in \Omega$$

iii) Mixing time

$$\tau_{\text{mix}}(i) = \min_n \{\|\nu K^n - \pi\|_{TV} \leq \epsilon, \forall \nu_0\}$$

We also have the following concepts to characterize the Markov chain.

Definition 7.1. The Burn-in period is the expected number of steps until the Markov chain enters the subspace of typical states. The subspace of typical states is a subspace of Ω where $\pi(x)$ is concentrated.

The burn-in notion is not very precise since it is hard to estimate when the distribution of the Markov chain νK^n is sufficiently close to the target distribution π , especially in high dimensional spaces.

Definition 7.2. The auto-correlation between the states of the Markov chain $\mathbf{x} = (x_0, \dots)$ is defined as

$$\text{Corr}(\tau) = \frac{1}{T} \sum_{t=t_0+1}^{t_0+T} (x_t - \bar{\mathbf{x}})(x_{t+\tau} - \bar{\mathbf{x}}), \quad \forall \tau \geq 0$$

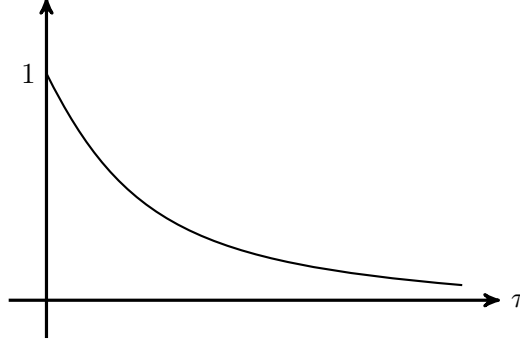


Figure 7.1: Usually the auto-correlation between samples decreases with the lag τ .

High auto-correlation implies slow convergence, and low auto-correlation implies fast convergence.

We can use the MC samples to integrate:

$$\theta = \int f(x)\pi(x)dx \cong \frac{1}{T} \sum_{t=t_0+1}^{t+T} f(x_t) = \hat{\theta}$$

$$\text{var}(\hat{\theta}) = E_{\text{samples}}[(\hat{\theta} - \theta)^2] = \frac{1}{m} \cdot \text{const}$$

where m is the effective number of independent samples.

7.1 Practical Methods for Monitoring

Deterministic algorithms converge to a point and convergence to that point can usually be monitored. For example in Maximum Likelihood Estimation (MLE) we can check the likelihood function $f(x)$ to see if the algorithm has converged.

In contrast, MCMC algorithms are stochastic, they converge to a distribution $\pi(x)$ and it is hard to determine whether convergence has occurred or not. However, there are some methods that can tell us something about convergence, such as:

1. Monitoring “sufficient statistics” of $\pi(x)$, such as the boundary length, total energy, etc. The statistics is averaged over space (a sample has many perturbations) or time (number of samples).

This way, we will project the state space Ω to the space of the sufficient statistics H . For any particular value of H , we have the inverse space

$$\Omega_{(h)} = \{x \in \Omega : H(x) = h\}$$

2. Run many Markov chains in parallel, starting from widely dispersed initial states, if possible, the extreme states. Monitoring can still be done using 1).

e.g. In Ising/Potts Model, we start from constant 0/1 (white/black), or white noise.

3. On top of 1)+2), we can monitor whether the MC (or MCs) forgot the past or initial points.
4. Monitoring the sampling temperature, T . In this respect we can:

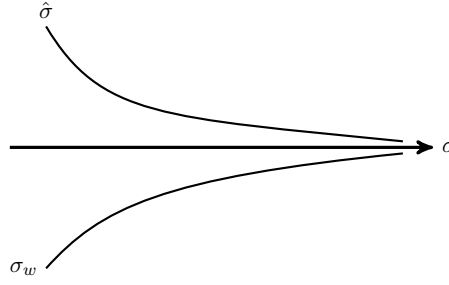


Figure 7.2: The $\hat{\sigma}$ and σ_w bound the true sigma. **IS THIS TRUE??**

- Monitor the rejection rate of the Metropolis-Hastings algorithm.
 - Monitor the entropy of the $\pi(x_i|x_{-i})$ in the Gibbs sampler (which reflects the temperature).
5. Suppose we have a state space Ω , we can simulate M different MC sequences, $\{\text{MC}_i; i = 1, 2, \dots, M\}$. We can compute the individual means ψ_i , and the total mean of all MCs, $\bar{\psi}$. Thus we have the between-MC variance and within-MC variance:

$$\sigma_w^2 = \frac{1}{M} \sum_{i=1}^M \sigma_i^2, \quad \sigma_i^2 = \frac{1}{T} \sum_{t=t_0}^{t_0+T} (x_i(t) - \psi_i)^2$$

$$\sigma_b^2 = \frac{1}{M-1} \sum_{i=1}^M (\psi_i - \bar{\psi})^2$$

Then we have an estimate of the variance of the MC:

$$\hat{\sigma} = \frac{T-1}{T} \sigma_w + \frac{1}{T} \sigma_B$$

7.2 Coupling Methods for Card shuffling

Shuffling a deck of cards is a Markov chain. We can use card shuffling to study coupling methods for Markov chains, where two or more MCs start independently and they slowly coalesce and move identically after a number of steps.

Suppose we have a deck with $n = 52$ cards. We can use Markov Chains to answer some questions such as: When are the cards thoroughly shuffled? Are all cards randomly distributed?

We have three ways to understand the problem.

1. Convergence is to a procedure, such as a shuffling process, since after each shuffling we get a new order.

By repeating this process N times, we get N decks (arrangements), and we can

- a) Test the distribution of the cards that appear at any location i , and compare with the empirical distribution of cards at i , which should be uniform.
 - b) Track the card that started at position i , and check its position distribution.
2. Check whether the new deck forgot the history, so players cannot cheat by memorizing the original order

3. We can monitor some marginal statistics between the cards, so that one cannot predict the next card based on cards in deck.

There are many ways to shuffle the cards, of which we present two here:

7.2.1 Shuffling to the top

This is a simple shuffling method easy to study theoretically. At each step a card i is selected at random and placed on the top of the deck. This way after many moves the deck is completely random.

To couple the MC associated to deck 1 with another MC associated to deck 2, we find the same card i in deck 2, and place it on the top. Thus, top cards are identical between deck 1, 2. This procedure is repeated until all 52 cards are picked at least once. This is also known as the “coupon collector’s problem”.

After some time T , the two decks are identical, have “coalesced”. The coalescence time T has the following characteristics

$$E[T] = n\left(\frac{1}{n} + \frac{1}{n-1} + \cdots + 1\right) \cong n \log n$$

$$\text{var}[T] \cong 0.72n$$

Remark 7.1. At each step we have to select among all n cards, so that the shuffling on deck one is unbiased otherwise it is no longer random.

Remark 7.2. Both decks are coupled by the same card i at each move.

7.2.2 Riffle shuffling

We divide the 52 cards into two decks according to $\text{Binomial}(n, \frac{1}{2})$. *i.e.* n Bernoulli($\frac{1}{2}$) trials. This way a number k of cards are in deck 1 and $n-k$ are in deck 2. The number k follows the distribution:

$$K \sim P(k) = \frac{1}{2^n} \binom{n}{k} = \frac{1}{2^n} \cdot \frac{n!}{k!(n-k)!}$$

This can be better understood by an inverse shuffling process, where “inverse” is similar to rewinding a video, *i.e.* playing video backwards.

At each shuffling, we simulate a binary bit for each card, $b_1, b_2, \dots, b_n \sim \text{Bernoulli}(\frac{1}{2})$, n times total. Then we go back and put all 0’s on top of 1’s.

After t shuffling operations, each card i from the original order is associated with t bits of code.

$$x_i = b_{1i}b_{2i} \cdots b_{ti}$$

When t is large so that all $\{x_i\}$ are distinct, then the order of the cards corresponds to the values of x_i , in the sense that the card that has a 0 last is above the other card in the deck. Then, the ordering after time t is completely random, since the order is dictated only by the bits x_i .

The number t will be equal to a classic statistical problem: What is the probability of dropping n balls in 2^t bins, so that each bin has at most 1 ball? The bins correspond to the 2^t bit combinations (b_1, b_2, \dots, b_t) without repetition and the balls correspond to the n cards.

7.3 Geometric Bounds, Bottleneck and Conductance

In this section, we introduce some key concepts about MCMC convergence rates without proofs.

7.3.1 Geometric convergence

Let (ν, K, Ω) be a Markov chain on a finite state space Ω with initial probability ν and transition kernel K . If K is irreducible and aperiodic, then

$$\|\nu \cdot K^n - \pi\|_{\text{TV}} \leq c \cdot r^n$$

where c is constant and $0 < r < 1$ is the geometric rate.

For such a Markov chain, there exists $n_0 > 0$ such that

$$K^{n_0}(x, y) > 0, \quad \forall x, y \in \Omega$$

There are many ways to prove it, since we do not have r specified.

- i) Using the contraction coefficient. The contraction coefficient for K is the maximum TV-norm between any two rows in the transition kernel and is calculated by

$$C(K) = \max_{x, y} \|K(x, \cdot) - K(y, \cdot)\|_{\text{TV}}$$

THIS PROOF IS NOT COMPLETE

- ii) Using Perron-Frobenius Theorem. Recall Theorem 3.4:

Theorem (Perron-Frobenius Theorem). *For any primitive (irreducible and aperiodic) $N \times N$ stochastic matrix K , K has eigenvalues*

$$1 = \lambda_1 > |\lambda_2| > \dots > |\lambda_r|$$

with multiplicities m_1, \dots, m_r and left and right eigenvectors $(\mathbf{u}_i, \mathbf{v}_i)$ respectively. Then $\mathbf{u}_1 = \pi, \mathbf{v}_1 = \mathbf{1}$, and

$$K^n = \mathbf{1} \cdot \pi' + O(n^{m_2-1} |\lambda_2|^n).$$

More specifically, we have the bound on each starting state $\mathbf{x}_0 \in \Omega$

$$\|\nu K^n - \pi\| \leq \sqrt{\frac{1 - \pi(\mathbf{x}_0)}{4\pi(\mathbf{x}_0)}} \cdot \lambda_{\text{slem}}^n$$

This bound is known as the Diaconis-Hanlon bound [54].

Now we need to analyze what factors bound λ_{slem} . λ_{slem} is often related to the worst case scenario (state, vertices, edge). There are two key concepts in the MCMC convergence literature: the “bottleneck” and “conductance” of a trade map.

Trade map (transition graph). Going back to the analogy of a Markov chain Kernel K with a trade between people (see Example 3.2), we define a graph $G = \langle V, E \rangle$ where $V = \Omega$ is the finite set of states, and

$$E = \{ \langle x, y \rangle; x, y \in V, K(x, y) > 0 \}$$

Each edge $e = \langle x, y \rangle$ is weighted by $Q(e) = \pi(x) \cdot K(x, y)$.

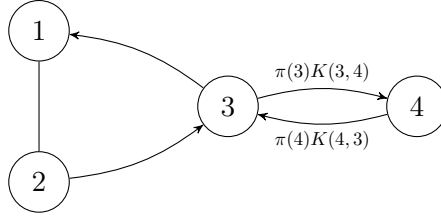


Figure 7.3: The graph associated to a MCMC.

Now we study some properties of this map to diagnose convergence. By irreducibility, $\forall x \neq y \in \Omega$, x and y are connected through many paths,

$$x \xrightarrow{\gamma} y$$

We define a weighted path

$$\Gamma_{xy} \stackrel{\text{def}}{=} (x, \dots, y)$$

has effective “length”:

$$\gamma_{xy} \stackrel{\text{def}}{=} \sum_{\langle s, t \rangle \in \Gamma_{xy}} \frac{1}{\pi(s)K(s, t)}$$

Thus, a lower probability to go from x to y will imply a longer path, and one needs to wait a long time to go from x to y .

Bottleneck. The connectivity of G can be measured by a “bottleneck” measure

$$\kappa = \max_{e \in E} \sum_{\Gamma_{xy} \ni e} \gamma_{xy} \cdot \pi(x)\pi(y)$$

where e^* is the bottleneck and κ is the measure. Intuitively, e^* will be like the “Panama Canal” or the “Strait of Magellan”, it will cause the traffic jam.

Poincaré inequality implies that:

$$\lambda_{\text{slem}} \leq 1 - \kappa^{-1}$$

Conductance of G . Suppose we divide the state space Ω into two subspaces.

$$\Omega = S \cup S^c$$

We define the transition probability,

$$K(S, S^c) \stackrel{\text{def}}{=} \sum_{x \in S} \sum_{y \in S^c} K(x, y)$$

Let $\pi(S) = \sum_{x \in S} \pi(x)$ as the capacity of S , and

$$Q(S, S^c) = \sum_{\substack{x \in S \\ y \in S^c}} \pi(x) \cdot K(x, y)$$

be the flow out of S .

Then the “conductance” of G is defined as

$$h \stackrel{\text{def}}{=} \min_{S: \pi(S) \leq \frac{1}{2}} \frac{Q(S, S^c)}{\pi(S)}$$

In the case of small conductance there exists S with large $\pi(S)$ but small $Q(S, S^c)$.

Cheeger's Inequality [55]:

$$1 - 2h \leq \lambda_{\text{slcm}} \leq 1 - \frac{h^2}{2}$$

These bounds are intuitive, but does not really guide the design in practice. In practice heuristics are used such as DDMCMC or the SW cut algorithm to speed up the Markov Chain.

7.4 Peskun's Ordering and Ergodicity Theorem

Now, we return to our early motivation for designing MCMC. Recall the Ergodicity theorem 3.5 from Chapter 3.

$$\theta = \int f(x)\pi(x)dx \cong \hat{\theta} = \frac{1}{n} \sum_{t=1}^n f(x^{(t)})$$

by samples $\{x^{(1)}, \dots, x^{(n)}\} \sim \pi(X)$ obtained by MCMC.

The efficiency of an MC is ultimately measured by the variance

$$\text{var}(\hat{\theta}) = \lim_{n \rightarrow \infty} \frac{1}{n} \text{var} \left\{ \sum_{t=1}^n f(x^{(t)}) \right\}$$

Suppose two MCs K_1 and K_2 have the same invariant probability π . We introduce a partial order among all such K 's

$$\Omega_\pi = \{K : \pi K = \pi, K \text{ irreducible and aperiodic}\}.$$

We say that K_1 dominates K_2 , *i.e.*, $K_1 \succeq K_2$, if $K_1(x, y) \geq K_2(x, y)$, $\forall x \neq y$.

Theorem 7.3 (Peskun). *If $K_1 \succeq K_2$ then $\text{var}(\hat{\theta}_1) \leq \text{var}(\hat{\theta}_2)$.*

Example 7.1. Consider the following two Markov chains:

MC1: $K_1(x, y) = Q(x, y) \cdot \alpha(x, y) = Q(x, y) \cdot \min \left(1, \frac{Q(y, x)\pi(y)}{Q(x, y)\pi(x)} \right)$ – Metropolis-Hastings design.

MC2: $K_2(x, y) = Q(x, y) \cdot \frac{\pi(y)Q(y, x)}{\pi(x)Q(x, y) + \pi(y)Q(y, x)}$ – Baker's design.

One can prove that $K_1 \succeq K_2$.

Example 7.2. Metropolized Gibbs sample \succeq Gibbs sampler.

7.5 Path Coupling and Exact Sampling

Example 7.3. We consider a $n \times n$ lattice (e.g. $n = 64$) and its graph consisting of the 4-nearest neighbors. The Ising model comes from physics where it models magnetic material on a lattice of charged particles with two possible spin states, -1 (down) or 1 (up). Let X be the spin states of the lattice, so the variable X_s at each site s is the spin state, taking values in $\{-1, 1\}$. The model for spin interaction of ferromagnetic material assigns positive interaction energy to spins of the opposite direction. Formally, the energy of the system is the Ising model:

$$H(X) = - \sum_{\langle s, t \rangle \in C} \beta X_s X_t,$$

where C are all the graph edges (4-nearest neighbors) of the lattice and β is interaction strength (possibly inhomogeneous).

This induces the probability measure for each possible state of the lattice:

$$P(X) = \frac{1}{Z} \exp^{-H(X)}$$

We simulate two Markov Chains with the Gibbs sampler:

MC1 starts with all sites being 1 (call it the white chain) and its state is denoted by X^1 ;

MC2 starts with all sites being 0 (call it the black chain) and its state is denoted by X^2 .

At each step, the Gibbs sampler picks up a site s in both images, and calculates the conditional probabilities, $p(X_s^1 | X_{\partial s}^1)$ and $p(X_s^2 | X_{\partial s}^2)$. It updates the variables X_s^1 and X_s^2 according to the above two conditional probabilities, and shares the same random number $r \in [0, 1]$ to sample the two variables. The two Markov Chains are said to be “coupled”.

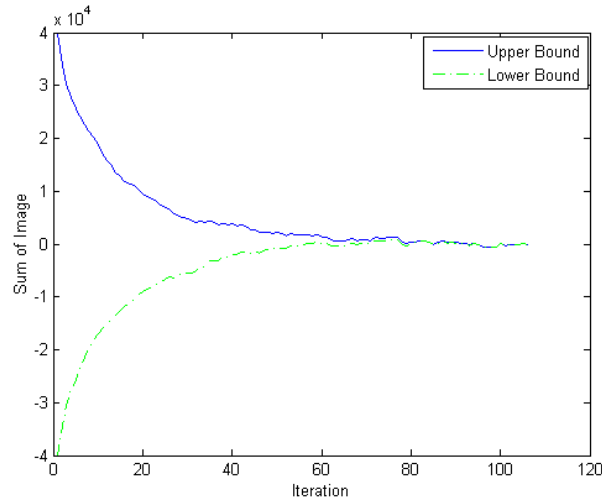


Figure 7.4: The total magnetization $\sum_s X_s$ on the Ising model with $\beta = 0.35$ for the MC1 and MC2, which coalesced at $\tau = 105$.

It can be proved that $X_s^1 \geq X_s^2, \forall s$ in any step. That is, the white chain is always above the black chain. An example for $\beta = 0.35$ is shown in Figure 7.4.

When the two chains meet each other, i.e. $X_s^1 = X_s^2, \forall s$ after many sweeps, they are said to “coalesce”. They will stay in the same state forever as they are driven by the same random number at each step. We denote the coalescence time (number of sweeps) by τ . The images after time τ are said to be exact samples from the Ising model.

The main tool and concepts for exact sampling:

1. Coupling
2. Simulation from the past
3. Monotonicity

7.5.1 Coupling from the past

The idea of coupling from the past is to run the simulation backward in time from each state, keeping track of the states where each chain ends up in. It is intuitive that once two states map to a single state after a simulation from time $-t$ to time 0, they will remain the same for simulations from $-t-1$ if the same randomness numbers are used.

Coupling from the past ensures that after a finite number of rounds of simulation M , the measure $\rho(i)$ of the state i we end up with is sufficiently close to the equilibrium distribution of the chain $\pi(i)$, i.e. $\|\rho(i) - \pi(i)\| < \epsilon$.

The output of a fixed-time backwards simulation is given by $F_{-M}^0(i)$, where $F_{t_1}^{t_2}$ is defined as the composition $f_{t_2-1} \circ f_{t_2-2} \circ \dots \circ f_{t_1+1} \circ f_{t_1}$.

1. Each $f_t(i)$ maps the state space to itself, with $-M \leq t \leq 1$
2. F_t^0 is updated via $F_t^0 = F_{t+1}^0 \circ f_t$
3. Coalescence is at the time point when F_t^0 becomes a constant map, $F_t^0(i) = F_t^0(i'), \forall i, i'$.

Theorem 7.4. *With probability 1 the coupling from the past procedure returns a value that is distributed according to the stationary distribution of the Markov chain.*

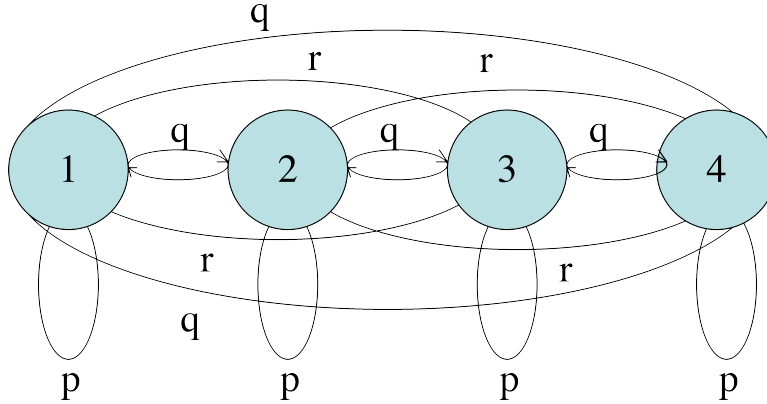


Figure 7.5: A Markov chain with four states, where $p = \frac{1}{3}$, $q = \frac{1}{4}$, $r = 1 - p - 2q$.

Example 7.4. Consider the four-state Markov chain shown in Figure 7.5. We simulate all of the states from the past. Coalescence happened after 5 iterations of simulation, as seen in Figure 7.6.

The state space S has a natural partial ordering, so that

$$x \leq y \Rightarrow \phi(x, u) \leq \phi(y, u),$$

where ϕ is the update rule, and u is the source of randomness.

The coalescence can be verified by tracking the maximal and minimal elements of the state space S .

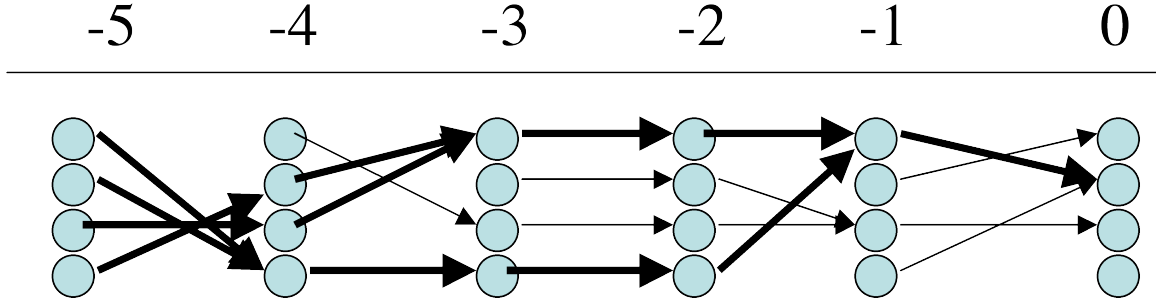


Figure 7.6: Coupling from the past for the MC from Figure 7.5. Coalescence happened after 5 iterations of simulation.

7.5.2 Application: sampling the Ising model

We now go back to the Ising model from example 7.3.

Problem: It is not trivial to sample the Ising model because of its high dimensionality.

Solution: We use Gibbs sampler to update the chain based on the conditional distribution of each particular spin of the lattice, $P(s/\partial_s)$, where ∂_s is the neighbor system of s .

It is very easy to sample directly from this distribution.

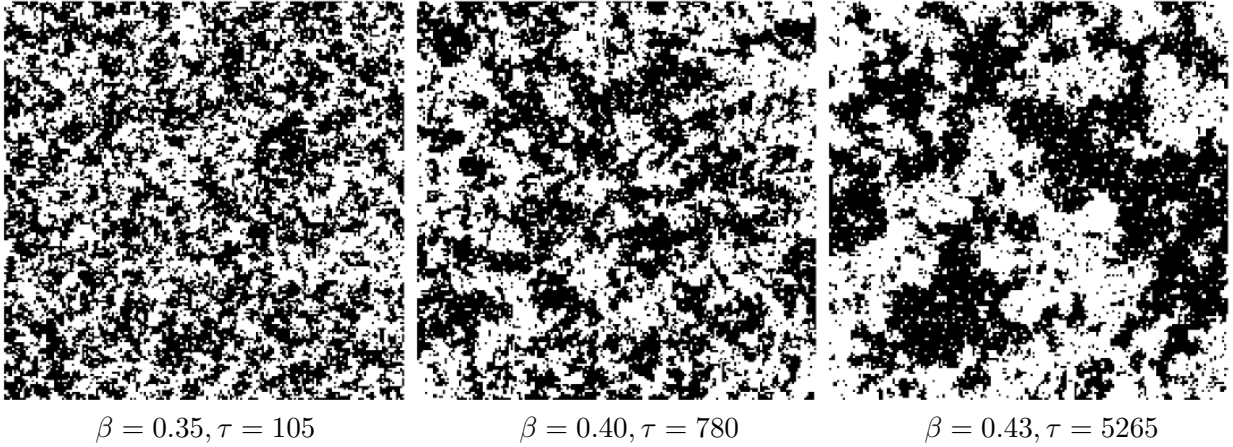


Figure 7.7: Samples of 2D ferromagnetic Ising model at different temperatures (different values of β). Lattice size: 200×200 .

It has been shown that if a deterministic (or semi-deterministic) scheme for updating all of the points in the lattice with the Gibbs sampler is employed, the induced Markov chain will converge to the joint distribution for the lattice, $P(I)$.

In Figure 7.7 are shown samples from the Ising model with different values of β at coalescence. Below each image is shown the value of β and the coalescence time τ .

In Figure 7.4 is shown the total magnetization $\sum_s X_s$ for the MC1 (white chain) and MC2 (black chain), for $\beta = 0.35$.

The energy at coalescence can vary for different random sequences, as shown in Figure 7.8.

In Figure 7.9 is shown the correlation

$$R(t) = \frac{1}{N} \sum_{i=1}^N \langle X^{(i)}, X^{(i+t)} \rangle$$

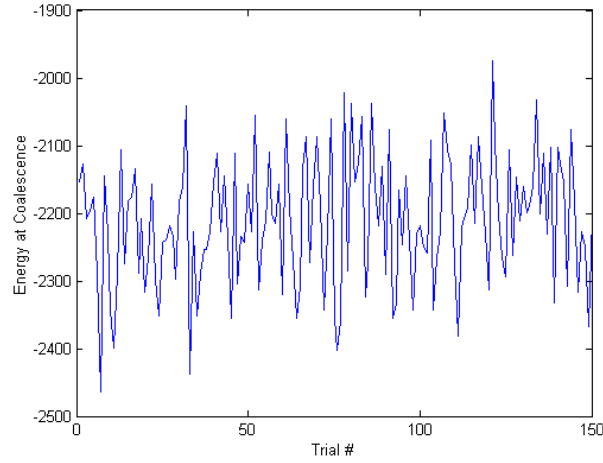


Figure 7.8: The energy at coalescence of 150 trials. $\beta = 0.40$, lattice size: 50×50 .

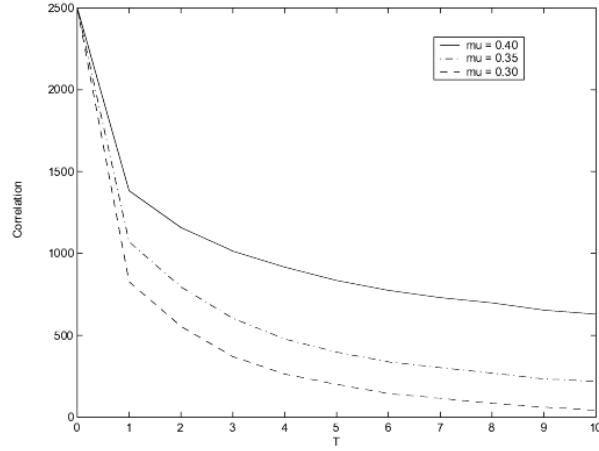


Figure 7.9: Correlation of equilibrium states at different temperatures. **LIMITS ARE WRONG**

between the states at different times i and time $i + t$.

If an external field X^{obs} is added to the model, the potential $H(X)$ becomes:

$$H(X) = - \sum_{\langle s,t \rangle \in C} \alpha X_s X_t - \sum_{\langle s \rangle \in I} X_s X_s^{obs}$$

The Ising model with external field can be used for image denoising, by using the observed noisy image as the external field X^{obs} and the sampled image X as the denoised image.

In Figure 7.10 is shown the obtained sampled image X by coupling from the past with the external field shown in top left, and different values of the interaction strength parameter β . The total magnetization for the upper and lower bound chains for $\beta = 1$ is shown in Figure 7.11.

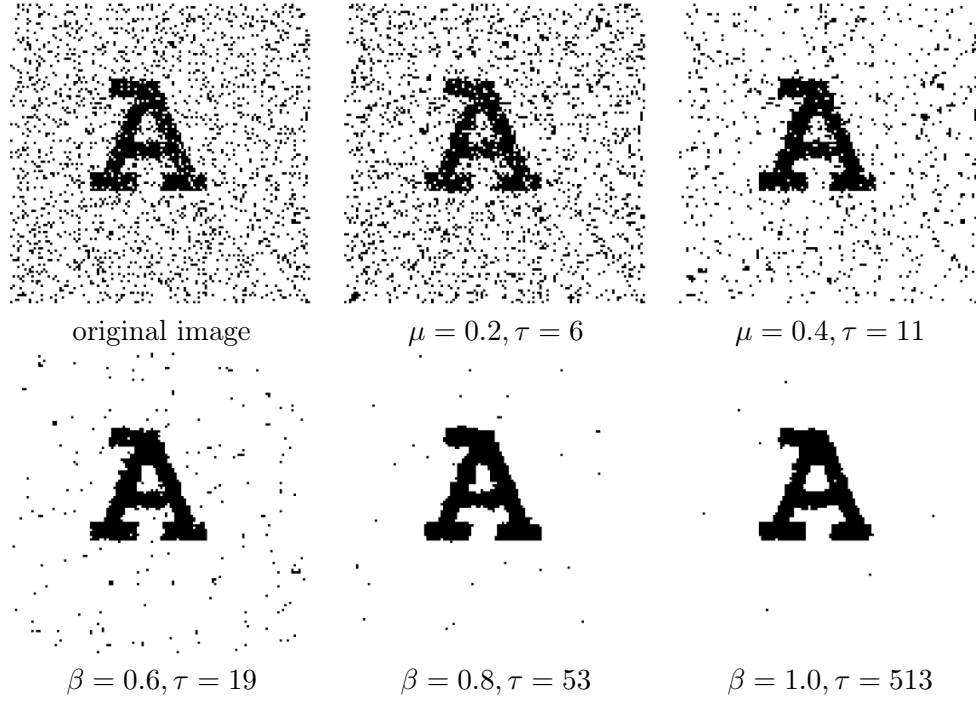


Figure 7.10: Simulation of the coupled Markov chains with external field for noise reduction.

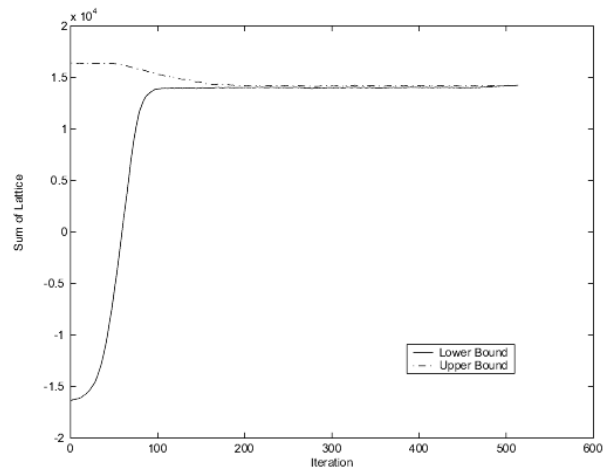


Figure 7.11: The upper bound and lower bound coalesced at $\tau = 513$, $\beta = 1$.

Chapter 8

Data Driven Markov Chain Monte Carlo

Image segmentation is a long standing problem in computer vision, and it is found difficult and challenging for two main reasons.

The first challenge is the fundamental complexity of modeling a vast amount of visual patterns that appear in generic images. The objective of image segmentation is to parse an image into its constituent components. The latter are various stochastic processes, such as attributed points, lines, curves, textures, lighting variations, and deformable objects. Thus a segmentation algorithm must incorporate many families of image models, and its performance is upper bounded by the accuracy of its image models.

The second challenge is the intrinsic ambiguities in image perception, especially when there is no specific task to guide the attention. Real world images are fundamentally ambiguous, and our perception of an image changes over time. Furthermore, an image often demonstrates details at multiple scales. Thus the more one looks at an image, the more one sees. Therefore, it must be wrong to think that a segmentation algorithm outputs only one result. In our opinion, image segmentation should be considered *a computing process* not a *vision task*. It should output multiple distinct solutions *dynamically* and *endlessly* so that these solutions “best preserve” the intrinsic ambiguity.

Motivated by the above two observations, we present a stochastic computing paradigm called *data-driven Markov chain Monte Carlo* (DDMCMC) for image segmentation. We proceed in five s.

Firstly, we formulate the problem in a Bayesian/MDL framework [104, 112, 224] with seven families of image models which compete to explain various visual patterns in an image, for example, flat regions, clutter, texture, smooth shading, etc.

Secondly, we decompose the solution space into an union of many subspaces of varying dimensions, and each subspace is a product of a number of subspaces for the image partition and image models (see Figure 8.3 for a space structure). The Bayesian posterior probability is distributed over such a heterogeneously structured space.

Thirdly, we design ergodic Markov chains to explore the solution space and sample the posterior probability. The Markov chains consist of two types of dynamics: jumps and diffusion. The jump dynamics simulate *reversible* split-and-merge, and model switching. The diffusion dynamics simulate boundary deformation, region growing, region competition [224], and model adaptation. We make the split and merge processes reversible, and the ergodicity and reversibility enable the algorithm to achieve nearly global optimal solution independent of initial segmentation conditions. Thus,

this demonstrates major progress over the previous region competition algorithm (Zhu and Yuille, 1996) [224].

Fourthly, we utilize data-driven techniques to guide the Markov chain search, and thus achieves tremendous speedup in comparison to previous MCMC algorithms [75,87,88]. In the literature, there are various techniques for improving the Markov chain speed, such as multi-resolution approaches [21,202], causal Markov models [21,152], and clustering [10,64,180,202]. In our DDMCMC paradigm, data-driven methods such as edge detection [29] and tracing, data clustering [33,40] are used. The results of these algorithms are expressed as weighted samples (or particles), which encode non-parametric probabilities in various subspaces. These probabilities respectively approximate the marginal probabilities of the Bayesian posterior probability, and they are used to design importance proposal probabilities to drive the Markov chains.

Fifthly, we propose a mathematical principle and a “*K*-adventurers” algorithm for selecting and pruning a set of *important* and *distinct* solutions from the Markov chain sequence and at multiple scales of details. The set of solutions encode an approximation to the Bayesian posterior probability. The multiple solutions are computed to minimize a Kullback-Leibler divergence from the approximate posterior to the true posterior, and they preserve the ambiguities in image segmentation.

In summary, the DDMCMC paradigm is about effectively creating particles (by bottom-up clustering/edge detection), composing particles (by importance proposals), and pruning particles (by a *K*-adventurers algorithm), and these particles represent hypotheses of various granularities in the solution space.

Conceptually, the DDMCMC paradigm also reveals the roles of some well-known segmentation algorithms. Algorithms such as split-and-merge, region growing, Snake [100] and balloon/bubble [174], region competition [224], and variational methods [104], and PDEs [156] can be viewed as various MCMC jump-diffusion dynamics with minor modifications. Other algorithms, such as edge detection [29] and clustering [40,52] compute importance proposal probabilities.

We test the algorithm on a wide variety of grey level and color images, and some results are shown in this section. We also demonstrate multiple solutions and verify the segmentation results by synthesizing (reconstructing) images through sampling the likelihood models.

8.1 Problem Formulation and Image Models

In this section, we formulate the problem in a Bayesian framework, and discuss the prior and likelihood models that are selected in our experiments.

8.1.1 The Bayesian Formulation for Segmentation

Let $\Lambda = \{(i, j) : 1 \leq i \leq L, 1 \leq j \leq H\}$ be an image lattice, and \mathbf{I}_Λ an image defined on Λ . For any point $v \in \Lambda$, $\mathbf{I}_v \in \{0, \dots, G\}$ is the pixel intensity for a grey level image, or $\mathbf{I}_v = (L_v, U_v, V_v)$ for a color image.¹ The problem of image segmentation refers to partitioning the lattice into an unknown number of K disjoint regions

$$\Lambda = \cup_{i=1}^K R_i, \quad R_i \cap R_j = \emptyset, \quad \forall i \neq j. \quad (8.1)$$

Each region $R \subset \Lambda$ needs not to be connected for reason of occlusion. We denote by $\Gamma_i = \partial R_i$ the boundary of R_i . As a slight complication, two notations are used interchangeably in the literature.

¹We transfer the (R,G,B) representation to (L^*, u^*, v^*) for better color distance measure.

One treats a region $R \in \Lambda$ as a discrete label map, and the other treats a region boundary $\Gamma(s) = \partial R$ as a continuous contour parameterized by s . The continuous representation is convenient for diffusions while the label map representation is better for maintaining the topology. The level set method [155, 156] provides a good transform between the two.

Each image region \mathbf{I}_R is supposed to be coherent in the sense that \mathbf{I}_R is a realization from a probabilistic model $p(\mathbf{I}_R; \Theta)$. Θ represents a stochastic process whose type is indexed by ℓ .

Thus a segmentation is denoted by a vector of hidden variables W , which describes the world state for generating the image \mathbf{I} .

$$W = (K, \{(R_i, \ell_i, \Theta_i); i = 1, 2, \dots, K\}).$$

In a Bayesian framework, we make inference about W from \mathbf{I} over a solution space Ω .

$$W \sim p(W|\mathbf{I}) \propto p(\mathbf{I}|W)p(W), \quad W \in \Omega.$$

As we mentioned before, the first challenge in segmentation is to obtain realistic image models. In the following, we briefly discuss the prior and likelihood models selected in our experiments.

8.1.2 The Prior Probability

The prior probability $p(W)$ is a product of the following four probabilities.

1. An exponential model for the number of regions $p(K) \propto e^{-\lambda_0 K}$.
2. A general smoothness Gibbs prior for the region boundaries $p(\Gamma) \propto e^{-\mu \oint_{\Gamma} ds}$.
3. A model for the size of each region. Recently both empirical and theoretical studies [3, 114] on the statistics of natural images indicate that the size of a region $A = |R|$ in natural images follows a distribution, $p(A) \propto \frac{1}{A^\alpha}$, $\alpha \sim 2.0$. Such a prior encourages large regions to form. In our experiments, we found this prior is not strong enough to enforce large regions, instead we take a distribution

$$p(A) \propto e^{-\gamma A^c}, \tag{8.2}$$

where $c = 0.9$ is a constant. γ is a scale factor which controls the scale of the segmentation. This scale factor is in spirit similar to the “clutter factor” found by Mumford and Gidas [146] in studying natural images. It is an indicator for how “busy” an image is. In our experiments, it is typically set to $\gamma = 2.0$ and is the only free parameter in this section.

4. The prior for the type of model $p(\ell)$ is assumed to be uniform, and the prior for the parameters Θ of an image model penalizes model complexity in terms of the number of parameters Θ , $p(\Theta|\ell) \propto e^{-\nu|\Theta|}$.

In summary, we have the following prior model

$$p(W) \propto p(K) \prod_{i=1}^K p(R_i) p(\ell_i) p(\Theta_i|\ell_i) \propto \exp\{-\lambda_0 K - \sum_{i=1}^K [\mu \oint_{\partial R_i} ds + \gamma |R_i|^c + \nu |\Theta_i|]\}.$$

8.1.3 The Likelihood for Grey Level Images

Visual patterns in different regions are assumed to be independent stochastic processes specified by $(\Theta_i, \ell_i), i = 1, 2, \dots, K$. Thus the likelihood is,²

$$p(\mathbf{I}|W) = \prod_{i=1}^K p(\mathbf{I}_{R_i}; \Theta_i, \ell_i).$$

The choice of models needs to balance model sufficiency and computational efficiency. In studying a large image set, we found that four types of regions appear most frequently in real world images. Figure 8.1 shows examples for the four types of regions in windows: a). flat regions with no distinct image structures, b). cluttered regions, c). regions with homogeneous textures, and d). inhomogeneous regions with globally smooth shading variations.

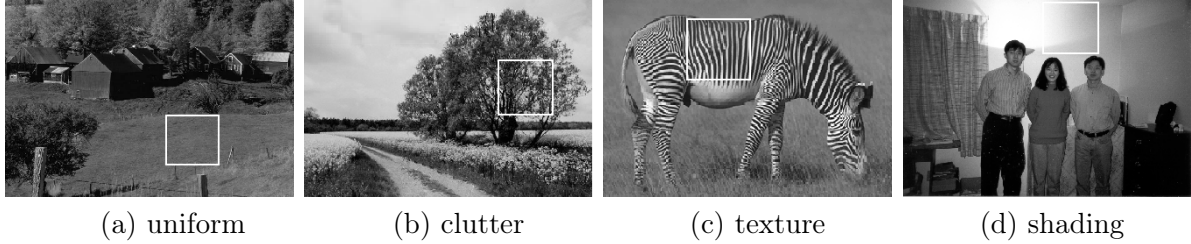


Figure 8.1: Four types of regions in the windows are typical in real world images.

We adopt the following four families of models for the four types of regions. The algorithm can switch between them by Markov chain jumps. The four families are indexed by $\ell \in \{g_1, g_2, g_3, g_4\}$ and denoted by ϖ_{g_1} , ϖ_{g_2} , ϖ_{g_3} , and ϖ_{g_4} respectively. Let $G(0; \sigma^2)$ be a Gaussian density centered at 0 with variance σ^2 .

1. Grey image model family $\ell = g_1$: ϖ_{g_1} . This assumes that pixel intensities in a region R are subject to independently and identically distributed (iid) Gaussian distribution,

$$p(\mathbf{I}_R; \Theta, g_1) = \prod_{v \in R} G(\mathbf{I}_v - \mu; \sigma^2), \quad \Theta = (\mu, \sigma) \in \varpi_{g_1}. \quad (8.3)$$

2. Grey image model family $\ell = g_2$: ϖ_{g_2} . This is a non-parametric intensity histogram $h(\cdot)$. In practice $h(\cdot)$ is discretized as a step function expressed by a vector (h_0, h_1, \dots, h_G) . Let n_j be the number of pixels in R with intensity level j .

$$p(\mathbf{I}_R; \Theta, g_2) = \prod_{v \in R} h(\mathbf{I}_v) = \prod_{j=0}^G h_j^{n_j}, \quad \Theta = (h_0, h_1, \dots, h_G) \in \varpi_{g_2}. \quad (8.4)$$

3. Grey image model family $\ell = g_3$: ϖ_{g_3} . This is a texture model FRAME [223] with pixel interactions captured by a set of Gabor filters. This family of models was demonstrated to be sufficient in realizing a wide variety of texture patterns. To facilitate the computation, we choose a set of 8 filters and formulate the model in pseudo-likelihood form [220]. The model is specified by a long vector $\Theta = (\beta_1, \beta_2, \dots, \beta_m) \in \varpi_{g_3}$, m is the total number of bins in the histograms of the 8

²As a slight notation complication, Θ, ℓ could be viewed as parameters or hidden variables in W . We use $p(\mathbf{I}; \Theta, \ell)$ in both situations for simplicity.

Gabor filtered images. Let ∂v denote the Markov neighborhood of $v \in R$, and $\mathbf{h}(\mathbf{I}_v|\mathbf{I}_{\partial v})$ the vector including 8 local histograms of filter responses in the neighborhood of pixel v . Each of the filter histogram counts the filter responses at pixels whose filter windows cover v . Thus we have

$$p(\mathbf{I}_R; \Theta, g_3) = \prod_{v \in R} p(\mathbf{I}_v|\mathbf{I}_{\partial v}; \Theta) = \prod_{v \in R} \frac{1}{Z_v} \exp\{-\langle \Theta, \mathbf{h}(\mathbf{I}_v|\mathbf{I}_{\partial v}) \rangle\}, \quad (8.5)$$

$\langle \cdot, \cdot \rangle$ is the inner product between two vectors, and the model is considered non-parametric. The reason for choosing the pseudo-likelihood expression is obvious: its normalizing constant can be computed exactly and Θ can be estimated easily from images. We refer to a recent paper [220] for discussions on the computation of this model and its variations, such as patch likelihood, etc.

4. Grey image model family g_4 : ϖ_{g_4} . The first three families of models are homogeneous, which fail in characterizing regions with shading effects, such as sky, lake, wall, perspective texture, etc. In the literature, such smooth regions are often modeled by low order Markov random fields, which again do not model the inhomogeneous pattern over space and often lead to over-segmentation. In our experiments, we adopt a 2D Bezier-spline model with sixteen equally spaced control points on Λ (i.e. we fix the knots). This is a generative type model. Let $B(x, y)$ be the Bezier surface, for any $v = (x, y) \in \Lambda$,

$$B(x, y) = U_{(x)}^T \times M \times U_{(y)}, \quad (8.6)$$

where $U_{(x)} = ((1-x)^3, 3x(1-x)^2, 3x^2(1-x), x^3)^T$ and $M = (m_{11}, m_{12}, m_{13}, m_{14}; \dots; m_{41}, \dots, m_{44})$. Therefore, the image model for a region R is,

$$p(\mathbf{I}_R; \Theta, g_4) = \prod_{v \in R} G(\mathbf{I}_v - B_v; \sigma^2), \quad \Theta = (M, \sigma) \in \varpi_{g_4}. \quad (8.7)$$

In summary, four types of models compete to explain a grey intensity region. Whoever fits the region better will have a higher likelihood. We denote by ϖ_{Θ}^g the grey level model space,

$$\Theta \in \varpi_{\Theta}^g = \varpi_{g_1} \cup \varpi_{g_2} \cup \varpi_{g_3} \cup \varpi_{g_4}.$$

8.1.4 Model calibration

The four image models should be calibrated for two reasons. Firstly, for computational efficiency, we prefer simple models with less parameters. However, penalizing the number of parameters is not enough in practice. When a region is of size over ~ 100 pixels, the data term dominates the prior and demands more complex models. Secondly, the pseudo-likelihood models in family ϖ_{g_3} are not a true likelihood as they depend on a rather big neighborhood, thus they are not directly comparable to the other three types of models.

To calibrate the likelihood probabilities, we did an empirical study. We collected a set of typical regions from natural images and manually divided them into four categories. For example, Figure 8.2 shows four typical images in the first column, which are cropped from the images in Figure 8.1. We denote the four images by $\mathbf{I}_i^{\text{obs}}, i = 1, 2, 3, 4$ on a lattice Λ_o . For each image $\mathbf{I}_i^{\text{obs}}$, we compute its per pixel coding length (minus log-likelihood) according to an optimal model within family ϖ_{g_j} computed by a maximum likelihood estimation for $j = 1, 2, 3, 4$.

$$L_{ij} = \min_{\varpi_{g_j} \ni \Theta} -\frac{\log p(\mathbf{I}_i^{\text{obs}}; \Theta, g_j)}{|\Lambda_o|}, \quad \text{for } 1 \leq i, j \leq 4. \quad (8.8)$$

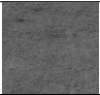


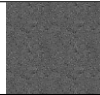




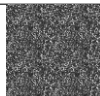

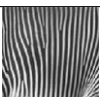


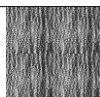






observed	ϖ_{g_1}	ϖ_{g_2}	ϖ_{g_3}	ϖ_{g_4}
				
$\mathbf{I}_1^{\text{obs}}$	$\mathbf{I}_{11}^{\text{syn}}$ $L_{11} = 11.957$	$\mathbf{I}_{12}^{\text{syn}}$ $L_{12} = 12.929$	$\mathbf{I}_{13}^{\text{syn}}$ $L_{13} = 13.680$	$\mathbf{I}_{14}^{\text{syn}}$ $L_{14} = 14.765$
				
$\mathbf{I}_2^{\text{obs}}$	$\mathbf{I}_{21}^{\text{syn}}$ $L_{21} = 13.503$	$\mathbf{I}_{22}^{\text{syn}}$ $L_{22} = 13.094$	$\mathbf{I}_{23}^{\text{syn}}$ $L_{23} = 12.749$	$\mathbf{I}_{24}^{\text{syn}}$ $L_{24} = 13.422$
				
$\mathbf{I}_3^{\text{obs}}$	$\mathbf{I}_{31}^{\text{syn}}$ $L_{31} = 13.852$	$\mathbf{I}_{32}^{\text{syn}}$ $L_{32} = 13.627$	$\mathbf{I}_{33}^{\text{syn}}$ $L_{33} = 12.514$	$\mathbf{I}_{34}^{\text{syn}}$ $L_{34} = 13.658$
				
$\mathbf{I}_4^{\text{obs}}$	$\mathbf{I}_{41}^{\text{syn}}$ $L_{41} = 13.121$	$\mathbf{I}_{42}^{\text{syn}}$ $L_{42} = 13.050$	$\mathbf{I}_{43}^{\text{syn}}$ $L_{43} = 11.259$	$\mathbf{I}_{44}^{\text{syn}}$ $L_{44} = 10.944$

Figure 8.2: Comparison study of four families of models. The first column is the original image regions cropped from four real world images shown in figure 8.1. The images in the 2-5 columns are synthesized images $\mathbf{I}_{ij}^{\text{syn}} \sim p(\mathbf{I}_R; \Theta_{ij}^*)$ sampled from the four families respectively each after an MLE fitting. The number below each synthesized image shows the per-pixel coding bits L_{ij} using each family of model.

We denote by $\Theta_{ij}^* \in \varpi_{g_j}$ optimal fit within each family and draw a typical sample (synthesis) from each fitted model,

$$\mathbf{I}_{ij}^{\text{syn}} \sim p(\mathbf{I}; \Theta_{ij}^*, g_j), \quad \text{for } 1 \leq i, j \leq 4.$$

We show $\mathbf{I}_i^{\text{obs}}$, $\mathbf{I}_{ij}^{\text{syn}}$, and L_{ij} in Figure 8.2 for $1 \leq i, j \leq 4$.

The results in Figure 8.2 show that the spline model has obviously the shortest coding length for the shading region, while the texture model fits the best for the three other regions. Then we choose to rectify these models by a constant factor e^{-c_j} for each pixel v ,

$$\hat{p}(\mathbf{I}_v; \Theta, g_j) = p(\mathbf{I}_v; \Theta, g_j) e^{-c_j} \quad \text{for } j = 1, 2, 3, 4.$$

$c_j, j = 1, 2, 3, 4$ are chosen so that the rectified coding length \hat{L}_{ij} reaches minimum when $i = j$. I.e. uniform regions, clutter regions, texture regions, and shading regions are best fitted by the models in ϖ_1 , ϖ_2 , ϖ_3 , and ϖ_4 respectively.

8.1.5 Image models for color

In experiments, we work on both grey level and color images. For color images, we adopt a (L^*, u^*, v^*) color space and adopted three families of models indexed by $\ell \in \{c_1, c_2, c_3\}$. Let $G(0; \Sigma)$ denote a 3D Gaussian density.

1. Color image model family c_1 : ϖ_{c_1} . This is an iid Gaussian model in (L^*, u^*, v^*) space.

$$p(\mathbf{I}_R; \Theta, c_1) = \prod_{v \in R} G(\mathbf{I}_v - \boldsymbol{\mu}; \Sigma), \quad \Theta = (\boldsymbol{\mu}, \Sigma) \in \varpi_{c_1}. \quad (8.9)$$

2. Color image model family c_2 : ϖ_{c_2} . This is a mixture of two Gaussians and is used for modeling textured color regions,

$$p(\mathbf{I}_R; \Theta, c_2) = \prod_{v \in R} [\alpha_1 G(\mathbf{I}_v - \boldsymbol{\mu}_1; \Sigma_1) + \alpha_2 G(\mathbf{I}_v - \boldsymbol{\mu}_2; \Sigma_2)].$$

Thus $\Theta = (\alpha_1, \boldsymbol{\mu}_1, \Sigma_1, \alpha_2, \boldsymbol{\mu}_2, \Sigma_2) \in \varpi_{c_2}$ are the parameters.

3. Color image model family c_3 : ϖ_{c_3} . We use three Bezier spline surfaces (see equation (8.6)) for L^* , u^* , and v^* respectively to characterize regions with gradually changing colors such as sky, wall, etc. Let $\mathbf{B}(x, y)$ be the color value in (L^*, u^*, v^*) space for any $v = (x, y) \in \Lambda$,

$$\mathbf{B}(x, y) = (U_{(x)}^T \times M_L \times U_{(y)}, U_{(x)}^T \times M_u \times U_{(y)}, U_{(x)}^T \times M_v \times U_{(y)})^T.$$

Thus the model is

$$p(\mathbf{I}_R; \Theta, c_3) = \prod_{v \in R} G(\mathbf{I}_v - \mathbf{B}_v; \Sigma),$$

where $\Theta = (M_L, M_u, M_v, \Sigma)$ are the parameters.

In summary, three types of models compete to explain a color region. Whoever fits the region better will have higher likelihood. We denote by ϖ_{Θ}^c the color model space, then

$$\varpi_{\Theta}^c = \varpi_{c_1} \cup \varpi_{c_2} \cup \varpi_{c_3}.$$

8.2 Anatomy of Solution Space

Before we design an algorithm, we need to study the structures of the solution space Ω in which the posterior probability $p(W|\mathbf{I})$ is distributed.

We start with the *partition space* for all possible partitions of a lattice Λ . When a lattice Λ is segmented into k disjoint regions, we call it a k -*partition* denoted by π_k ,

$$\pi_k = (R_1, R_2, \dots, R_k), \quad \cup_{i=1}^k R_i = \Lambda, \quad R_i \cap R_j = \emptyset, \quad \forall i \neq j. \quad (8.10)$$

If all pixels in each region are connected, then π_k is a connected component partition [202]. The set of all k -partitions, denoted by ϖ_{π_k} , is a quotient space of the set of all possible k -colorings divided by a permutation group \mathcal{PG} for the labels.

$$\varpi_{\pi_k} = \{(R_1, R_2, \dots, R_k) = \pi_k; \quad |R_i| > 0, \quad \forall i = 1, 2, \dots, k\} / \mathcal{PG}. \quad (8.11)$$

Thus we have a general partition space ϖ_{π} with the number of regions $1 \leq k \leq |\Lambda|$,

$$\varpi_{\pi} = \cup_{k=1}^{|\Lambda|} \varpi_{\pi_k}.$$

Then the solution space for W is a union of subspaces Ω_k , and each Ω_k is a product of one k -partition space ϖ_{π_k} and k spaces for the image models

$$\Omega = \cup_{k=1}^{|\Lambda|} \Omega_k = \cup_{k=1}^{|\Lambda|} [\varpi_{\pi_k} \times \underbrace{\varpi_{\Theta} \times \dots \times \varpi_{\Theta}}_k], \quad (8.12)$$

where $\varpi_{\Theta} = \cup_{i=1}^4 \varpi_{g_i}$ for grey level images, and $\varpi_{\Theta} = \cup_{i=1}^3 \varpi_{c_i}$ for color images.

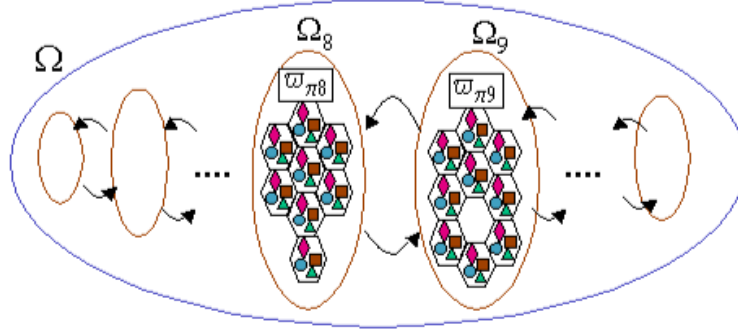


Figure 8.3: The anatomy of the solution space. The arrows represent Markov chain jumps, and the reversible jumps between two subspace Ω_8 and Ω_9 realize a split-and-merge of a region.

Figure 8.3 illustrates the structures of the solution space. In Figure 8.3, the four image families $\varpi_\ell, \ell = g_1, g_2, g_3, g_4$ are represented by the triangles, squares, diamonds and circles respectively. $\varpi_\Theta = \varpi_\Theta^g$ is represented by a hexagon containing the four shapes. The partition space ϖ_{π_k} is represented by a rectangle. Each subspace Ω_k consists of a rectangle and k hexagons, and each point $W \in \Omega_k$ represents a k -partition plus k image models for k regions.

We call Ω_k the **scene spaces**. ϖ_{π_k} and $\varpi_\ell, \ell = g_1, g_2, g_3, g_4$ (or $\ell = c_1, c_2, c_3$) are the basic components for constructing Ω and thus are called the **atomic spaces**. Sometimes we call ϖ_π a **partition space** and $\varpi_\ell, \ell = g_1, g_2, g_3, g_4, c_1, c_2, c_3$ the **cue spaces**.

8.3 Exploring the Solution Space by Ergodic Markov chains

The solution space in Figure 8.3 is typical for vision problems. The posterior probability $p(W|\mathbf{I})$ not only has an enormous number of local maxima but is distributed over subspaces of varying dimensions. To search for globally optimal solutions in such spaces, we adopt the Markov chain Monte Carlo (MCMC) techniques.

8.3.1 Three basic criteria for Markov chain design.

There are three basic requirements for Markov chain design.

Firstly, the Markov chain should be ergodic. That is, from an arbitrary initial segmentation $W_o \in \Omega$, the Markov chain can visit any other states $W \in \Omega$ in finite time. This disqualifies all greedy algorithms. Ergodicity is ensured by the jump-diffusion dynamics [88]. *Diffusion* realizes random moves within a subspace of fixed dimensions. *Jumps* realize reversible random walks between subspaces of different dimensions, as shown by the arrows in Figure 8.3.

Secondly, the Markov chain should be aperiodic. This is ensured by using the dynamics at random.

Thirdly, the Markov chain has stationary probability $p(W|\mathbf{I})$. This is replaced by a stronger condition of *detailed balance equations* which demands that every move should be reversible [87, 88]. All jumps in this section satisfy detailed balance and reversibility.

8.3.2 Five Markov chain dynamics

We adopt five types of Markov chain dynamics which are used at random with probabilities $p(1), \dots, p(5)$ respectively. The dynamics 1-2 are diffusion, and dynamics 3-5 are reversible jumps.

Dynamics 1: boundary diffusion/competition. For mathematical convenience, we switch to a continuous boundary representation for regions $R_i, i = 1, \dots, K$. These curves evolve to maximize the posterior probability through a region competition equation [224]. Let Γ_{ij} be the boundary between $R_i, R_j, \forall i, j$, and Θ_i, Θ_j the models for the two regions respectively. The motion of points $\Gamma_{ij}(s) = (x(s), y(s))$ follows the steepest ascent equation of the $\log p(W|\mathbf{I})$ plus a Brownian motion dB along the curve normal direction $\vec{n}(s)$. By variational calculus, this is [224],

$$\frac{d\Gamma_{ij}(s)}{dt} = [f_{prior}(s) + \log \frac{p(\mathbf{I}(x(s), y(s)); \Theta_i, \ell_i)}{p(\mathbf{I}(x(s), y(s)); \Theta_j, \ell_j)} + \sqrt{2T(t)}dB] \vec{n}(s).$$

The first two terms are derived from the prior and likelihood respectively. The Brownian motion is a normal distribution whose magnitude is controlled by a temperature $T(t)$ which decreases with time t . The Brownian motion helps to avoid local small pitfalls. The log-likelihood ratio requires that the image models are comparable. Dynamics 1 realizes diffusion within the atomic (or partition) space ϖ_{π_k} (i.e. moving within a rectangle of Figure 8.3).

Dynamics 2: model adaptation. This is simply to fit the parameters of a region by steepest ascent. One can add a Brownian motion, but it does not make much a difference in practice.

$$\frac{d\Theta_i}{dt} = \frac{\partial \log p(\mathbf{I}_{R_i}; \Theta_i, \ell_i)}{\partial \Theta_i}.$$

This realizes diffusion in the atomic (or cue) spaces $\varpi_l, \ell \in \{g_1, g_2, g_3, g_4, c_1, c_2, c_3\}$ (move within a triangle, square, diamond, or circle of Figure 8.3).

Dynamics 3-4: split and merge. Suppose at a certain time step, a region R_k with model Θ_k is split into two regions R_i and R_j with models Θ_i, Θ_j , or vice verse, and this realizes a jump between two states W to W' as shown by the arrows in Figure 8.3.

$$W = (K, (R_k, \ell_k, \Theta_k), W_-) \longleftrightarrow (K+1, (R_i, \ell_i, \Theta_i), (R_j, \ell_j, \Theta_j), W_-) = W',$$

where W_- denotes the remaining variables that are unchanged during the move. By the classic Metropolis-Hastings method [139], we need two proposal probabilities $G(W \rightarrow dW')$ and $G(W' \rightarrow dW)$. $G(W \rightarrow dW')$ is a conditional probability for how likely the Markov chain *proposes* to move to W' at state W , and $G(W' \rightarrow dW)$ is the proposal probability for coming back. The proposed split is then accepted with probability

$$\alpha(W \rightarrow dW') = \min(1, \frac{G(W' \rightarrow dW)p(W'|\mathbf{I})dW'}{G(W \rightarrow dW')p(W|\mathbf{I})dW}).$$

There are two routes (or “pathways” in a psychology language) for computing the split proposal $G(W \rightarrow dW')$.

In route 1, it first chooses a split move with probability $q(3)$, then chooses region R_k from a total of K regions at random, we denote this probability by $q(R_k)$. Given R_k , it chooses a candidate splitting boundary Γ_{ij} within R_k with probability $q(\Gamma_{ij}|R_k)$. Then for the two new regions R_i, R_j it chooses two new model types ℓ_i and ℓ_j with probabilities $q(\ell_i)$ and $q(\ell_j)$ respectively. Then it chooses $\Theta_i \in \varpi_{\ell_i}$ with probability $q(\Theta_i|R_i, \ell_i)$ and chooses Θ_j with probability $q(\Theta_j|R_j, \ell_j)$. Thus,

$$G(W \rightarrow dW') = q(3)q(R_k)q(\Gamma_{ij}|R_k)q(\ell_i)q(\Theta_i|R_i, \ell_i)q(\ell_j)q(\Theta_j|R_j, \ell_j)dW'. \quad (8.13)$$

In route 2, it first chooses two new region models Θ_i and Θ_j , and then decides the boundary Γ_{ij} . Thus,

$$G(W \rightarrow dW') = q(3)q(R_k)q(\ell_i)q(\ell_j)q(\Theta_i, \Theta_j|R_k, \ell_i, \ell_j)q(\Gamma_{ij}|R_k, \Theta_i, \Theta_j)dW'. \quad (8.14)$$

We shall discuss in later subsection that either of the two routes can be more effective than the other depending on the region R_k .

Similarly we have the merge proposal probability,

$$G(W' \rightarrow dW) = q(4)q(R_i, R_j)q(\ell_k)q(\Theta_k|R_k, \ell_k)dW. \quad (8.15)$$

$q(R_i, R_j)$ is the probability of choosing to merge two regions R_i and R_j at random.

Dynamics 5: switching image models. This switches the image model within the four families (three for color images) for a region R_i . For example, from texture description to a spline surface etc.

$$W = (\ell_i, \Theta_i, W_-) \longleftrightarrow (\ell'_i, \Theta'_i, W_-) = W'.$$

The proposal probabilities are

$$G(W \rightarrow dW') = q(5)q(R_i)q(\ell'_i)q(\Theta'_i|R_i, \ell'_i)dW', \quad (8.16)$$

$$G(W' \rightarrow dW) = q(5)q(R_i)q(\ell_i)q(\Theta_i|R_i, \ell_i)dW. \quad (8.17)$$

8.3.3 The bottlenecks

The speed of a Markov chain depends critically on the design of its proposal probabilities in the jumps. In our experiments, the proposal probabilities, such as $q(1), \dots, q(5)$, $q(R_k)$, $q(R_i, R_j)$, $q(\ell)$ are easy to specify and do not influence the convergence significantly. The real bottlenecks are caused by two proposal probabilities in the jump dynamics.

1. $q(\Gamma|R)$ in eqn.(8.13): Where is a good Γ for splitting a given region R ? $q(\Gamma|R)$ is a probability in the atomic (or partition) space ϖ_π .
2. $q(\Theta|R, \ell)$ in eqns (8.13), (8.15) and (8.17): For a given region R and a model family $\ell \in \{g_1, \dots, g_4, c_1, c_2, c_3\}$, what is a good Θ ? $q(\Theta|R, \ell)$ is a probability in the atomic (cue) space ϖ_ℓ .

It is worth mentioning that both probabilities $q(\Gamma|R)$ and $q(\Theta|R, \ell)$ cannot be replaced by deterministic decisions which were used in region competition [224] and others [112]. Otherwise, the Markov chain will not be reversible and thus reduce to a greedy algorithm. On the other hand, if we choose uniform distributions, it is equivalent to blind search, and the Markov chain will experience exponential “waiting” time before each jump. In fact, the length of the waiting time is proportional to the volume of the cue spaces. The design of these probabilities need to strike a balance between *speed* and *robustness* (non-greediness).

While it is hard to analytically derive a convergence rate for complicated algorithms that we are dealing with, it is revealing to observe the following theorem in a simple case [138]

Theorem 8.1. Sampling a target density $p(x)$ by independence Metropolis-Hastings algorithm with proposal probability $q(x)$. Let $P^n(x_o, y)$ be the probability of a random walk to reach point y at n s. If there exists $\rho > 0$ such that,

$$\frac{q(x)}{p(x)} \geq \rho, \quad \forall x,$$

then the convergence measured by a L_1 norm distance

$$||P^n(x_o, \cdot) - p|| \leq (1 - \rho)^n.$$

This theorem states that the proposal probability $q(x)$ should be very close to $p(x)$ for fast convergence. In our case, $q(\Gamma|R)$ and $q(\Theta|R, \ell)$ should be equal to the conditional probabilities of some marginal probabilities of the posterior $p(W|\mathbf{I})$ within the atomic spaces ϖ_π and ϖ_ℓ respectively. That is,

$$q_\Gamma^*(\Gamma_{ij}|R_k) = p(\Gamma_{ij}|\mathbf{I}, R_k), \quad q_\Theta^*(\Theta|R, \ell) = p(\Theta|\mathbf{I}, R, \ell), \quad \forall \ell. \quad (8.18)$$

Unfortunately, q_Γ^* and q_Θ^* have to integrate information from the entire image \mathbf{I} , and thus are intractable. We must seek approximations, and this is where the data-driven methods step in.

In the next section, we discuss data clustering for each atomic space ϖ_ℓ , $\ell \in \{c_1, c_2, c_3\}$ and $\ell \in \{g_1, g_2, g_3, g_4\}$ and edge detection in ϖ_π . The results of clustering and edge detection are expressed as non-parametric probabilities for approximating the ideal marginal probabilities q_Γ^* and q_Θ^* in these atomic space respectively.

8.4 Data-Driven Methods

8.4.1 Method I: clustering in atomic spaces

Given an image \mathbf{I} (grey or color) on lattice Λ , we extract a feature vector F_v^ℓ at each pixel $v \in \Lambda$. The dimension of F_v^ℓ depends on the image model indexed by ℓ . Then we have a collection of vectors

$$\mathcal{U}^\ell = \{F_v^\ell : v \in \Lambda\}.$$

In practice, v can be subsampled for computational ease. The set of vectors are clustered by either an EM method [51] or a mean-shift clustering [33, 40] algorithm to \mathcal{U}^ℓ . The EM-clustering approximates the points density in \mathcal{U}^ℓ by a mixture of m Gaussians, and it extends from the m -mean clustering by a soft cluster assignment to each vector F_v . The mean-shift algorithm assumes a non-parametric distribution for \mathcal{U}^ℓ and seeks the modes (local maxima) in its density (after some Gaussian window smoothing). Both algorithms return a list of m weighted clusters $\Theta_1^\ell, \Theta_2^\ell, \dots, \Theta_m^\ell$ with weights $\omega_i^\ell, i = 1, 2, \dots, m$, and we denote by

$$\mathcal{P}^\ell = \{(\omega_i^\ell, \Theta_i^\ell) : i = 1, 2, \dots, m\}. \quad (8.19)$$

We call $(\omega_i^\ell, \Theta_i^\ell)$ a *weighted atomic (or cue) particle* in ϖ_ℓ for $\ell \in \{c_1, c_3, g_1, g_2, g_3, g_4\}$.³ The size m is chosen to be conservative, or it can be computed in a coarse-to-fine strategy with a limit $m = |\mathcal{U}^\ell|$. This is well discussed in the literature [33, 40].

In the clustering algorithms, each feature F_v^ℓ and thus its location v is classified to a cluster Θ_i^ℓ with probability $S_{i,v}^\ell$,

$$S_{i,v}^\ell = p(F_v^\ell; \Theta_i^\ell), \quad \text{with} \quad \sum_{i=1}^m S_{i,v}^\ell = 1, \quad \forall v \in \Lambda, \quad \forall \ell.$$

This is a soft assignment and can be computed by the distance from F_v to the cluster centers. We call

$$S_i^\ell = \{S_{i,v}^\ell : v \in \Lambda\}, \quad \text{for } i = 1, 2, \dots, m, \quad \forall \ell \quad (8.20)$$

a **saliency map** associated with cue particle Θ_i^ℓ .

In the following, we discuss each model family with experiments.

³The atomic space ϖ_{c_2} is a composition of two ϖ_{c_1} , and thus is computed from ϖ_{c_1} .

Computing cue particles in ϖ_{c_1}

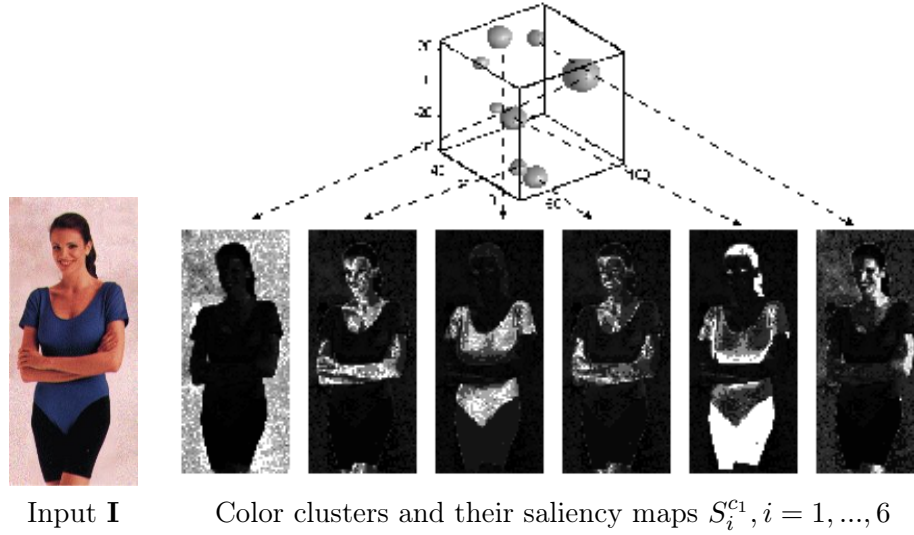


Figure 8.4: A color image and its clusters in (L^*, u^*, v^*) space for ϖ_{c_1} , the second row are six of the saliency maps associated with the color clusters.

For color images, we take $F_v = (L_v, U_v, V_v)$ and apply a mean-shift algorithm [33, 40] to compute color clusters in ϖ_{c_1} . For example, Figure 8.4 shows a few color clusters (balls) in a cubic $((L^*, u^*, v^*)$ -space) for a simple color image (left), the size of the balls represents the weights $\omega_i^{c_1}$. Each cluster is associated with a saliency map $S_i^{c_1}$ for $i = 1, 2, \dots, 6$ in the second row, and the bright areas mean high probabilities. From left to right are respectively, background, skin, shirt, shadowed skin, pant and hair, highlighted skin.

Computing cue particles in ϖ_{c_3}

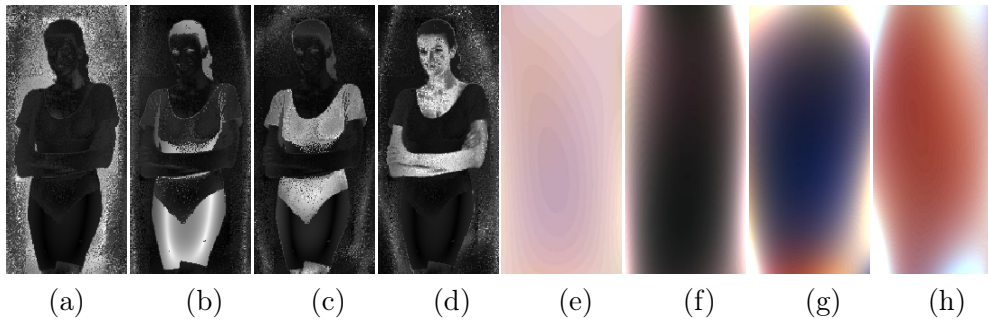


Figure 8.5: (a)-(d) are saliency maps associated with four clusters in ϖ_{c_3} . (e)-(h) are the color spline surfaces for the four clusters.

Each point v contributes its color $\mathbf{I}_v = (L_v, U_v, V_v)$ as “surface heights”, and we apply an EM-clustering to find the spline surface models. Figure 8.5 shows the clustering result for the woman image. Figures 8.5.a-d are saliency maps $S_i^{c_3}$ for $i = 1, 2, 3, 4$. Figures 8.5.e-h are the four reconstructed images according to fitted spline surfaces which recover some global illumination variations.

Computing cue particles in ϖ_{g_1}

In this model, the feature space $F_v = \mathbf{I}_v$ is simply the intensity, and \mathcal{U}^{g_1} is the image intensity histogram. We simply apply a mean-shift algorithm to get the modes (peaks) of the histogram and the breadth of each peak decides its variance.

Figure 8.6 shows six saliency maps $S_i^{g_1}, i = 1, 2, \dots, 6$ for a zebra image (the original image is shown in Figure 8.14.a). In the clustering map on the left in Figure 8.6, each pixel is assigned to its most likely particle.

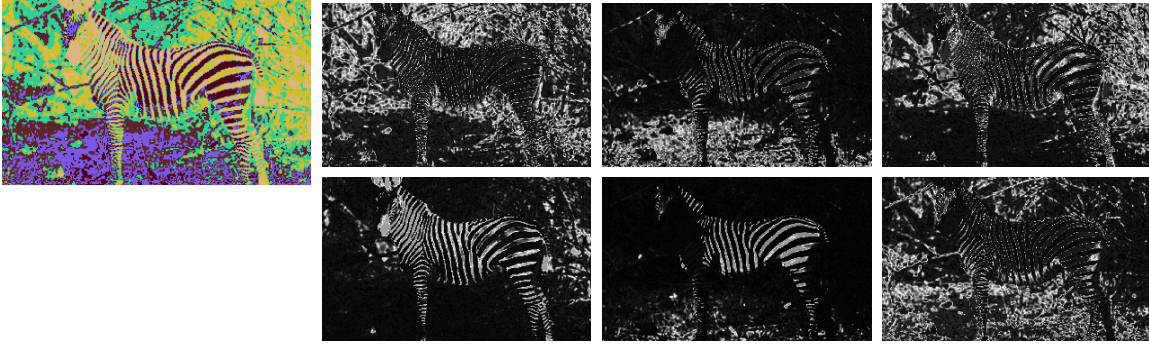


Figure 8.6: A clustering map (left) for ϖ_{g_1} and six saliency maps $S_i^{g_1}, i = 1, \dots, 6$ of a zebra image (input is in Fig. 8.14.a).

Computing the cue particles in ϖ_{g_2}

For clustering in ϖ_{g_2} , at each subsampled pixel $v \in \Lambda$, we compute F_v as a local intensity histogram $F_v = (h_{v0}, \dots, h_{vG})$ accumulated over a local window centered at v . Then an EM clustering is applied to compute the cue particles, and each particle $\Theta_i^{g_2}, i = 1, \dots, m$ is a histogram. This model is used for clutter regions.

Figure 8.7 shows the clustering results on the same zebra image.

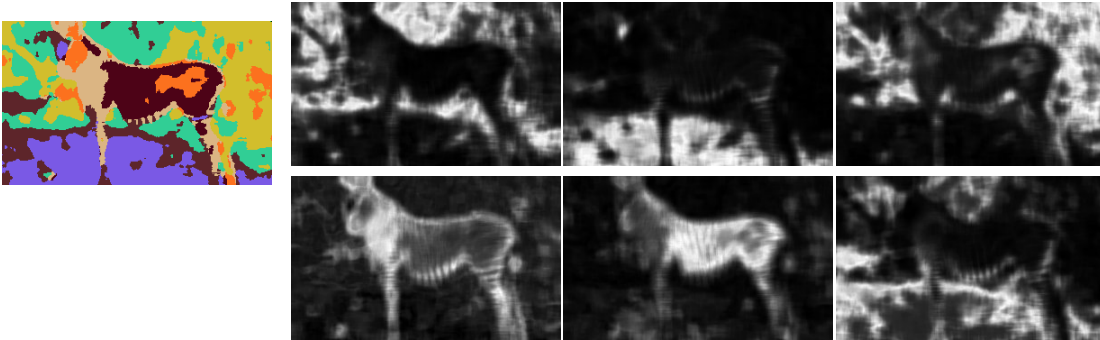


Figure 8.7: A clustering map (left) for ϖ_{g_2} and six saliency maps $S_i^{g_2}, i = 1..6$ of a zebra image (input is in Fig. 8.14.a).

Computing cue particles ϖ_{g_3}

At each subsampled pixel $v \in \Lambda$, we compute a set of 8 local histograms for 8 filters over a local window of 12×12 pixels. We choose 8 filters for computational convenience: one δ filter, two gradient filters, one Laplacian of Gaussian filter, and four Gabor filters. Each histogram has 9 bins. Then $F_v^{g_3} = (h_{v,1,1}, \dots, h_{v,8,9})$ is the feature. An EM clustering is applied to find the m mean histograms $\bar{\mathbf{h}}_i, i = 1, 2, \dots, m$. We can compute the cue particles for texture models $\Theta_i^{g_3}$ from $\bar{\mathbf{h}}_i$ for $i = 1, 2, \dots, m$. A detailed account of this transform is referred to a previous paper [220].

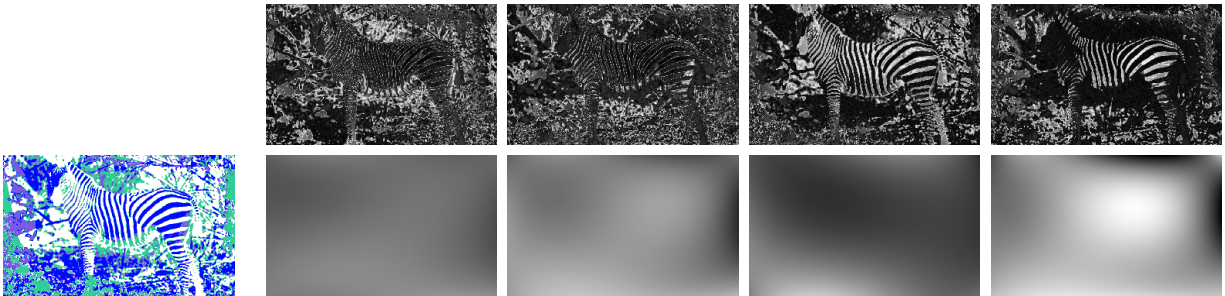


Figure 8.8: Texture clustering. A clustering map (left) and four saliency maps for four particles $\Theta_i^{g_3}, i = 1, 2, \dots, 4$.

Figure 8.8 shows the texture clustering results on the zebra image with one clustering map on the left, and five saliency maps for five particles $\Theta_i^{g_3}, i = 1, 2, \dots, 5$.

Computing cue particles in ϖ_{g_4}

Each point v contributes its intensity $\mathbf{I}_v = F_v$ as a “surface height”, and we apply an EM-clustering to find the spline surface models. Figure 8.9 shows a clustering result for the zebra image with four surfaces. The second row shows the four surfaces which recover some global illumination variations. Unlike the texture clustering results which capture the zebra strips as a whole region, the surface models separate the black and white stripes as two regions – another valid perception. Interestingly, the black and white strips in the zebra skin both have shading changes which are fitted by the spline models.



A clustering map

Four saliency maps and surfaces

Figure 8.9: Clustering result on the zebra image under Bezier surface model. The left image is the clustering map. The first row of images on the right side are the saliency maps. The second row shows the fitted surfaces using the surface height as intensity.

8.4.2 Method II: Edge detection

We detect intensity edges using Canny edge detector [29] and color edges using a method in [115], and trace edges to form a partition of the image lattice. We choose edges at three scales according to edge strength, and thus compute the partition maps in three coarse-to-fine scales. We choose

not to discuss the details, but show some results using the two running examples: the woman and zebra images.

Figure 8.10.a shows a color image and three scales of partitions. Since this image has strong color cue, the edge maps are very informative about where the region boundaries are. In contrast, the edge maps for the zebra image are very messy, as Figure 8.11 shows.

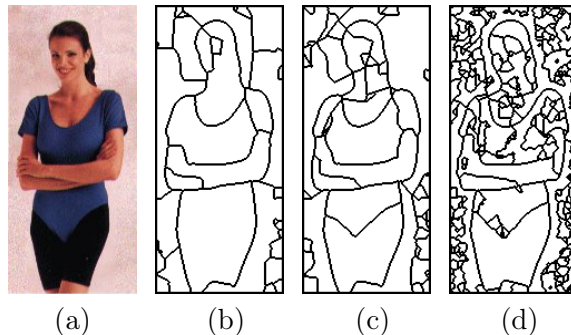


Figure 8.10: Partition maps at three scales of details for a color image. (a) Input image. (b) Partition map at scale 1. (c) Partition map at scale 2. (d) Partition map at scale 3.



Figure 8.11: A grey level image and three partition maps at three scales. (a) Input image. (b) Partition map at scale 1. (c) Partition map at scale 2. (d) Partition map at scale 3.

8.5 Computing importance proposal probabilities

It is generally acknowledged in the community that clustering and edge detection algorithms can sometimes produce good segmentations or even perfect results for some images, but very often they are far from being reliable for generic images, as the experiments in Figures 8.4-8.11 demonstrate. It is also true that sometimes one of the image models and edge detection scales could do a better job in segmenting some regions than other models and scales, but we do not know a priori what types of regions present in a generic image. Thus we compute all models and edge detection at multiple scales and then utilize the clustering and edge detection results probabilistically. MCMC theory provides a framework for integrating these probabilistic information in a principled way under the guidance of a globally defined Bayesian posterior probability.

We explain how the the importance proposal probabilities $q(\Theta|R, \ell)$ and $q(\Gamma_{ij}|R_k)$ in Section 8.3.3. are computed from the data-driven results.

Computing importance proposal probability $q(\Theta|R, \ell)$

The clustering method in an atomic (cue) space ϖ_ℓ outputs a set of weighted cue particles \mathcal{P}^ℓ . \mathcal{P}^ℓ encodes a non-parametric probability in ϖ_ℓ ,

$$q(\Theta|\Lambda, \ell) = \sum_{i=1}^m \omega_i^\ell G(\Theta - \Theta_i^\ell), \quad \text{with} \quad \sum_{i=1}^m \omega_i^\ell = 1, \quad (8.21)$$

where $G(x)$ is a Parzen window centered at 0. As a matter of fact, $q(\Theta|\Lambda, \ell) = q(\Theta|\mathbf{I})$ is an approximation to a marginal probability of the posterior $p(W|\mathbf{I})$ on cue space $\varpi_\ell, \ell \in \{g_1, g_2, g_3, g_4, c_1, c_3\}$, since the partition π is integrated out in EM-clustering.

$q(\Theta|\Lambda, \ell)$ is computed once for the whole image, and $q(\Theta|R, \ell)$ is computed from $q(\Theta|\Lambda, \ell)$ for each R at run time. It proceeds in the following. Each cluster $\Theta_i^\ell, i = 1, 2, \dots, m$ receives a real-valued vote from the pixel $v \in R$ in region R , and the accumulative vote is the summation of the saliency map S_i^ℓ associated with Θ_i^ℓ , i.e.,

$$p_i = \frac{1}{|R|} \sum_{v \in R} S_{i,v}^\ell, \quad i = 1, 2, \dots, m, \quad \forall \ell.$$

Obviously the clusters which receive high votes should have high chance to be chosen. Thus we sample a new image model Θ for region R ,

$$\Theta \sim q(\Theta|R, \ell) = \sum_{i=1}^m p_i G(\Theta - \Theta_i^\ell). \quad (8.22)$$

Equation (8.22) explains how we choose (or propose) an image model for a region R . We first draw a cluster i at random according to probability $p = (p_1, p_2, \dots, p_m)$, and then do a random perturbation at Θ_i^ℓ . Thus any $\Theta \in \varpi_\ell$ has a non-zero probability to be chosen for robustness and ergodicity. Intuitively the clustering results with local votes propose the “hottest” portions of the space in a probabilistic way to guide the jump dynamics.

In practice, one could implement a multi-resolution (on a pyramid) clustering algorithm over smaller local windows, thus the clusters $\Theta_i^\ell, i = 1, 2, \dots, m$ will be more effective at the expense of some overhead computing.

Computing importance proposal probability $q(\Gamma|R)$

By edge detection and tracing, we obtain partition maps denoted by $\Delta^{(s)}$ at multiple scales $s = 1, 2, 3$. In fact, each partition map $\Delta^{(s)}$ consists of a set of “meta-regions” $r_i^{(s)}, i = 1, 2, \dots, n$,

$$\Delta^{(s)}(\Lambda) = \{r_i^{(s)} : i = 1, 2, \dots, n, \cup_{i=1}^n r_i^{(s)} = \Lambda\}, \quad \text{for } s = 1, 2, 3.$$

These meta regions are then used in combination to form $K \leq n$ regions $R_1^{(s)}, R_2^{(s)}, \dots, R_K^{(s)}$,

$$R_i^{(s)} = \cup_j r_j^{(s)}, \quad \text{with } r_j^{(s)} \in \Delta^{(s)}, \quad \forall i = 1, 2, \dots, K.$$

One could put a constraint that all meta-regions in a region $R_i^{(s)}$ are connected.

Let $\pi_k^{(s)} = (R_1^{(s)}, R_2^{(s)}, \dots, R_k^{(s)})$ denote a k -partition based on $\Delta^{(s)}$. $\pi_k^{(s)}$ is different from the general k -partition π_k because regions $R_i^{(s)}, i = 1, \dots, K$ in $\pi_k^{(s)}$ are limited to the meta-regions. We denote by $\Pi_k^{(s)}$ the set of all k -partitions based on a partition map $\Delta^{(s)}$.

$$\Pi_k^{(s)} = \{(R_1^{(s)}, R_2^{(s)}, \dots, R_k^{(s)}) = \pi_k^{(s)} : \cup_{i=1}^k R_i^{(s)} = \Lambda\}. \quad (8.23)$$

We call each $\pi_k^{(s)}$ in $\Pi_k^{(s)}$ a *k-partition particle* in atomic (partition) space ϖ_{π_k} . Like the clusters in a cue space, $\Pi_k^{(s)}$ is a sparse subset of ϖ_{π_k} , and it narrows the search in ϖ_{π_k} to the most promising portions.

So each partition map $\Delta^{(s)}$ encodes a probability in the atomic (partition) space ϖ_{π_k} .

$$q^{(s)}(\pi_k) = \frac{1}{|\Pi_k^{(s)}|} \sum_{j=1}^{|\Pi_k^{(s)}|} G(\pi_k - \pi_{k,j}^{(s)}), \quad \text{for } s = 1, 2, 3. \quad \forall k. \quad (8.24)$$

$G(\cdot)$ is a smooth window centered at 0 and its smoothness accounts for boundary deformations and forms a cluster around each partition particle, and $\pi_k - \pi_{k,j}^{(s)}$ measures the difference between two partition maps π_k and $\pi_{k,j}^{(s)}$. Martin et al. [136] recently proposed a method of measuring such difference and we use a simplified version. In the finest resolution, all meta regions reduce to pixels, and $\Pi_k^{(s)}$ is then equal to the atomic space ϖ_{π_k} . We adopt equal weights for all partitions $\pi_k^{(s)}$, and one may add other geometric preferences to some partitions.

In summary, the partition maps at all scales encode a non-parametric probability in ϖ_{π_k} ,

$$q(\pi_k) = \sum_s q(s) q^{(s)}(\pi_k), \quad \forall k.$$

This $q(\pi_k)$ can be considered as an approximation to the marginal posterior probability $p(\pi_k|\mathbf{I})$.

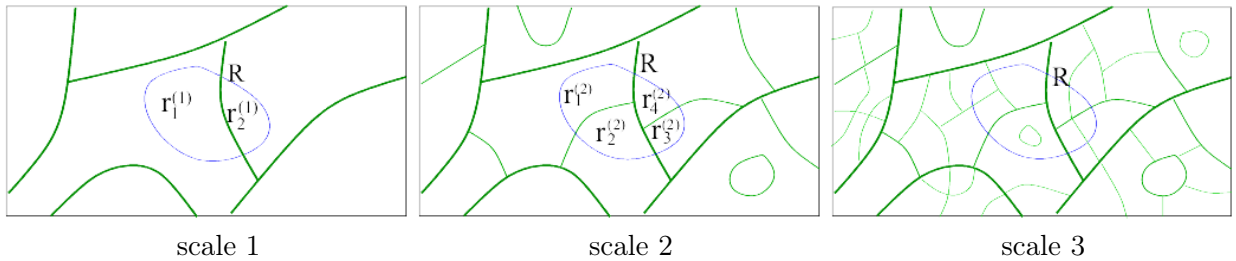


Figure 8.12: A candidate region R_k is superimposed on the partition maps at three scales for computing a candidate boundary Γ_{ij} for the pending split.

The partition maps $\Delta^{(s)}, \forall s$ (or $q(\pi_k), \forall k$ implicitly) are computed once for the whole image, then the importance proposal probability $q(\Gamma|R)$ is computed from $q(\pi_k)$ for each region as a conditional probability at run time, like in the cue spaces.

Figure 8.12 illustrates an example. We show partition maps $\Delta^{(s)}(\Lambda)$ at three scales, and the edges are shown at width 3, 2, 1 respectively for $s = 1, 2, 3$. A candidate region R is proposed to split. $q(\Gamma|R)$ is the probability for proposing a splitting boundary Γ .

We superimpose R on the three partition maps. The intersections between R and the meta regions generate three sets

$$\Delta^{(s)}(R) = \{r_j^{(s)} : r_j^{(s)} = R \cap r_j \text{ for } r_j \in \Delta^{(s)}(\Lambda), \text{ and } \cup_i r_i^{(s)} = R\}, \quad s = 1, 2, 3.$$

For example, in Figure 8.12, $\Delta^{(1)}(R) = \{r_1^{(1)}, r_2^{(1)}\}$, $\Delta^{(2)}(R) = \{r_1^{(2)}, r_2^{(2)}, r_3^{(2)}, r_4^{(2)}\}$, and so on.

Thus we can define $\pi_c^{(s)}(R) = (R_1^{(s)}, R_2^{(s)}, \dots, R_c^{(s)})$ as a c -partition of region R based on $\Delta^{(s)}(R)$, and define a c -partition space of R as

$$\Pi_c^{(s)}(R) = \{(R_1^{(s)}, R_2^{(s)}, \dots, R_c^{(s)}) = \pi_c^{(s)}(R) : \cup_{i=1}^c R_i^{(s)} = R\}, \quad \forall s. \quad (8.25)$$

We can define distributions on $\Pi_c^{(s)}(R)$.

$$q^{(s)}(\pi_c(R)) = \frac{1}{|\Pi_c^{(s)}(R)|} \sum_{j=1}^{|\Pi_c^{(s)}(R)|} G(\pi_c - \pi_{c,j}^{(s)}(R)), \quad \text{for } s = 1, 2, 3, \quad \forall c. \quad (8.26)$$

Thus one can propose to split R into c pieces, in a general case,

$$\pi_c(R) \sim q(\pi_c(R)) = \sum_s q(s) q^{(s)}(\pi_c(R)).$$

That is, we first select a scale s with probability $q(s)$. $q(s)$ depends on R . For example, for a large region R we can choose coarse scale with higher probability, and choose a fine scale for small regions. Then we choose a c -partition from the set $\Pi_c^{(s)}(R)$. In our implementation, $c = 2$ is chosen as a special case for easy implementation. It is trivial to show that an arbitrary c -partition of region R , $\pi_c(R)$, can be generated through composing $\pi_2(R)$ in multiple s . Obviously there is a big overhead for choosing large c .

Computing $q(\Theta_i, \Theta_j | R, \ell_i, \ell_j)$ and $q(\Gamma_{ij} | R, \Theta_i, \Theta_j)$

In some cases, we find the second route useful for splitting a region which we discussed in designing MCMC dynamics 3-4 (see equation (8.14)).

For example, there are two ways to perceive the zebra in Figure 8.14. One perceives the zebra as one textured region (by a model in ϖ_{g_3}). The other sees it as one region of black stripes plus one region of white strips and thus uses two models in ϖ_{g_1} or ϖ_{g_4} . The Markov chain should be able to switch between the two perceptions effectively (see results in Figure 8.14.b-d). This is necessary and typical for the transitions between any texture regions and intensity regions.

Because the number of strips in such textures is large, the first split procedure (route 1) is very ineffective, and it works on one strip at a time. This motivates the second pathway for split dynamics.

For a candidate region R , we first propose two new region models (we always assume the same labels $\ell_i = \ell_j$), this can be done by twice sampling the importance proposal probabilities $q(\Theta | R, \ell)$, so

$$(\Theta_i, \Theta_j) \sim q(\Theta_i, \Theta_j | R, \ell_i, \ell_j) = q(\Theta_i | R, \ell_i) q(\Theta_j | R, \ell_j).$$

Obviously we exclude Θ_i from the candidate set when we select Θ_j . Then we decide on the boundary Γ $q(\Gamma_{ij} | R, \Theta_i, \Theta_j)$ by randomly labeling the pixels in R according to probabilities of the saliency maps.

A unifying framework

To summarize this section, the DDMCMC paradigm provides a unifying framework for understanding the roles of many existing image segmentation algorithms. Firstly, edge detection and tracing

methods [29, 115] compute implicitly a marginal probability $q(\pi|\mathbf{I})$ on the partition space ϖ_π . Secondly, clustering algorithms [33, 40] compute a marginal probability on the model space ϖ_ℓ for various models ℓ . Thirdly, the split-and-merge and model switching [10] realize jump dynamics. Fourthly, region growing and competition methods [156, 224] realize diffusion dynamics for evolving the region boundaries.

8.6 Computing Multiple Distinct Solutions

8.6.1 Motivation and a mathematical principle

The DDMMCMC paradigm samples solutions from the posterior $W \sim p(W|\mathbf{I})$ endlessly. To extract an optimal result, one can take an annealing strategy and use the conventional *maximum a posteriori* (MAP) estimator

$$W^* = \arg \max_{W \in \Omega} p(W|\mathbf{I}).$$

In this section we argue that it is desirable and often critical to have the ability of computing multiple distinct solutions for the following reasons.

Firstly, natural scenes are intrinsically ambiguous, and for an image \mathbf{I} many competing organizations and interpretations exist in visual perception.

Secondly, for robustness, decisions should be left to the last stage of computation when a segmentation process is integrated with a specific task. Therefore it is best to maintain a set of *typical* solutions.

Thirdly, preserving multiple solutions is necessary when the prior and likelihood models are not perfect. Because the globally optimal solution may not be semantically more meaningful than some other inferior local maxima.

However, simply keeping a set of samples from the Markov chain sequence is not enough, because it often collects a set of segmentations that are trivially different from each other. Here we present a mathematical principle for computing important and distinctive solutions in the space Ω , relying on the techniques presented in Section 2.5 for preserving sample diversity in importance sampling.

Let $S = \{(\omega_i, W_i) : i = 1, \dots, K\}$ be a set of K weighted solutions which we call “scene particles”, with weights their posterior probabilities $\omega_i = p(W|\mathbf{I})$, $i = 1, 2, \dots, K$. (Note that there is a slight abuse of notation, we use K for the number of regions in W before. Here it is a different K). S encodes a non-parametric probability in Ω ,

$$\hat{p}(W|\mathbf{I}) = \sum_{i=1}^K \frac{\omega_i}{\omega} G(W - W_i), \quad \sum_{i=1}^K \omega_i = \omega.$$

G is a Gaussian window in Ω .

As all image ambiguities are captured in the Bayesian posterior probability, to reflect the intrinsic ambiguities, we should compute the set of solutions S that best preserve the posterior probability. Thus we let $\hat{p}(W|\mathbf{I})$ approach $p(W|\mathbf{I})$ by minimizing a Kullback-Leibler divergence $D(p||\hat{p})$ under a complexity constraint $|S| = K$,

$$S^* = \arg \min_{|S|=K} D(p||\hat{p}) = \arg \min_{|S|=K} \int p(W|\mathbf{I}) \log \frac{p(W|\mathbf{I})}{\hat{p}(W|\mathbf{I})} dW. \quad (8.27)$$

This criterion extends the conventional MAP estimator.

8.6.2 A K -adventurers algorithm for multiple solutions

Fortunately, the KL-divergence $D(p||\hat{p})$ can be estimated fairly accurately by a distance measure $\hat{D}(p||\hat{p})$ which is computable, thanks to two observations of the posterior probability $p(W|\mathbf{I})$ which has many separable modes. The idea is simple. We can always represent $p(W|\mathbf{I})$ by a mixture of Gaussian, i.e. a set of N particles with N large enough. By ergodicity, the Markov chain is supposed to visit these significant modes over time! Thus our goal is to extract K distinct solutions from the Markov chain sampling process.

Here we present a greedy algorithm for computing S^* approximately. We call the algorithm – “ K -adventurers algorithm.”⁴

Suppose we have a set of K particles S at step t . At time $t + 1$, we obtain a new particle (or a number of particles) by MCMC, usually following a successful jump. We augment the set S to S_+ by adding the new particle(s). Then we eliminate one particle (or a number of particles) from S_+ to get a new S_{new} by minimizing the approximative KL divergence, divergence $\hat{D}(p_+||p_{\text{new}})$.

The k -adventurers algorithm

1. Initializing S and \hat{p} by repeating one initial solution K times.
2. Repeat
3. Compute a new particle $(\omega_{K+1}, \mathbf{x}_{K+1})$ by DDMCMC after a successful jump.
4. $S_+ \leftarrow S \cup \{(\omega_{K+1}, \mathbf{x}_{K+1})\}$.
5. $\hat{p} \leftarrow S_+$.
6. For $i = 1, 2, \dots, K + 1$ do
7. $S_{-i} \leftarrow S_+ / \{(\omega_i, \mathbf{x}_i)\}$.
8. $\hat{p}_{-i} \leftarrow S_{-i}$.
9. $d_i = D(p||\hat{p}_{-i})$.
10. $i^* = \arg \min_{i \in \{1, 2, \dots, K+1\}} d_i$.
11. $S \leftarrow S_{-i^*}$, $\hat{p} \leftarrow \hat{p}_{-i^*}$

In practice, we run multiple Markov chains and add new particles to the set S in a batch fashion.

8.7 Image Segmentation Experiments

The DDMCMC paradigm was tested extensively on many grey level, color, and textured images. This section shows some examples and more are available on our website⁵. It was also tested in a benchmark dataset of 50 natural images in both color and grey level [136] by the Berkeley group⁶, where the results by DDMCMC and other methods such as [175] are displayed in comparison to those by a number of human subjects. Each tested algorithm uses the same parameter setting for all the benchmark images and thus the results were obtained purely automatically.

We first show our working example on the color woman image. Following the importance proposal probabilities for the edges in Figure 8.10 and for color clustering in Figure 8.4, we simulated three Markov chains with three different initial segmentations shown in Figure 8.13 (top row). The energy changes $(-\log p(W|\mathbf{I}))$ of the three MCMCs are plotted in Figure 8.13 against time s . Figure 8.13 shows two different solutions W_1, W_2 obtained by a Markov chain using K -adventurers algorithm. To verify the computed solution W_i , we synthesized an image by sampling from the likelihood $\mathbf{I}_i^{\text{syn}} \sim p(\mathbf{I}|W_i)$, $i = 1, 2$. The synthesis is a good way to examine the sufficiency of models in segmentation.

Figure 8.14 shows three segmentations on a grey level zebra image. As we discussed before, the DDMCMC algorithm in this section has only one free parameter γ which is a “clutter factor” in

⁴The name follows a statistics metaphor told by Mumford to S.C. Zhu. A team of K adventurers want to occupy K largest islands in an ocean while keeping apart from each other’s territories.

⁵See <http://vcla.stat.ucla.edu/old/Segmentation/Segment.htm>

⁶See www.cs.berkeley.edu/~dmartin/segbench/BSDS100/html/benchmark

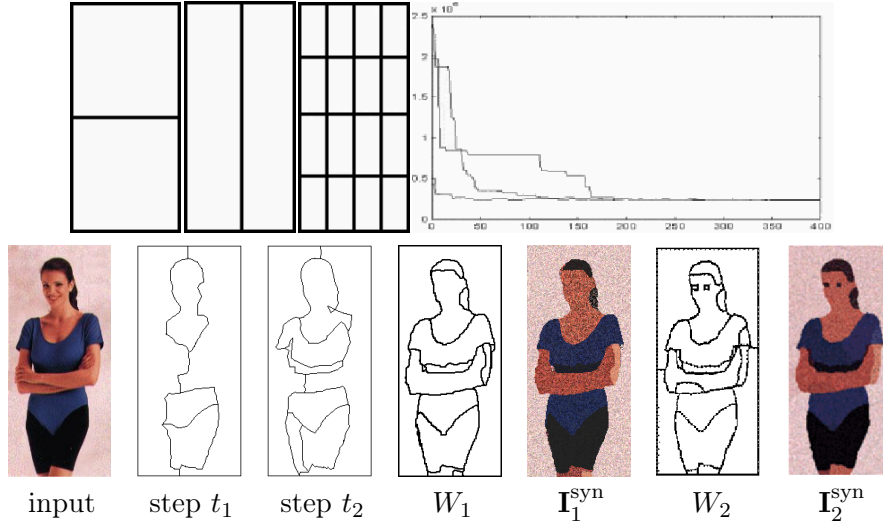


Figure 8.13: Segmenting a color image by DDMCMC with two solutions. See text for explanation.

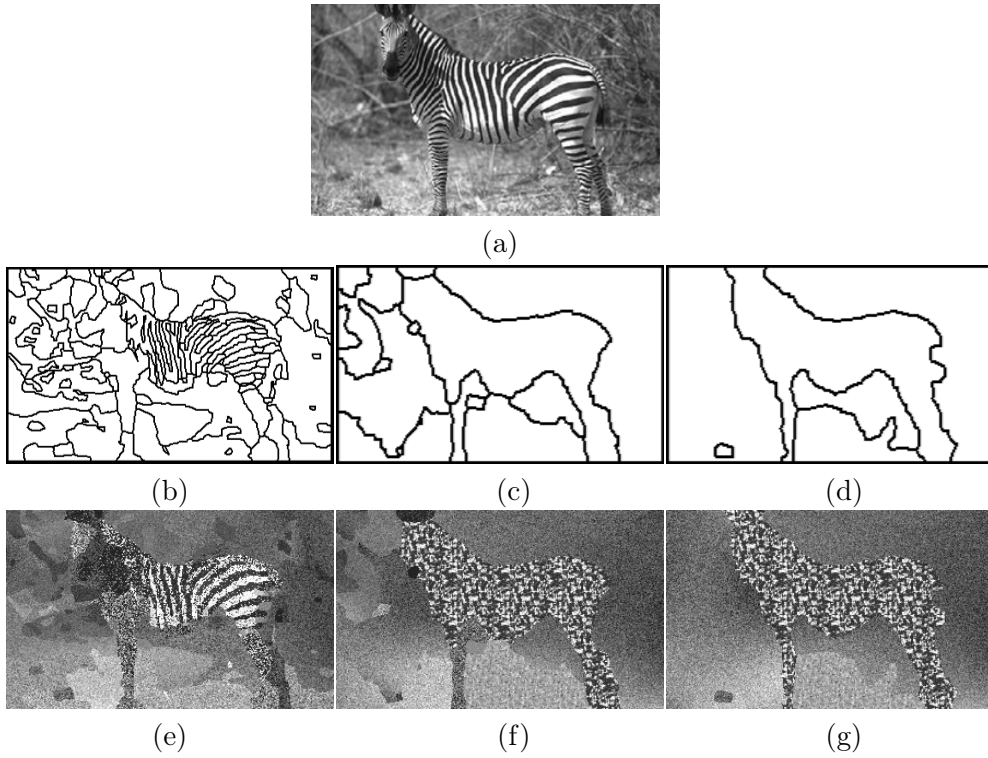


Figure 8.14: Experiments on the grey level zebra image with three solutions. (a) input image. (b)-(d) are three solutions, $W_i, i = 1, 2, 3$, for the zebra image. (e)-(g) are synthesized images $I_i^{\text{syn}} \sim p(\mathbf{I}|\mathbf{W}_i^*)$ for verifying the results.

the prior model (See equation (8.2)). It controls the extents of segmentations. A big γ encourages coarse segmentation with large regions. We normally extract results at three scales by setting $\gamma = 1.0, 2.0, 3.0$ respectively. In our experiments, the K -adventurers algorithm is effective only for computing distinct solutions in a certain scale. We expect the parameter γ can be fixed to a constant if we form an image pyramid with multiple scales and conduct segmentation with K -adventurers

algorithm at each scale, and then propagate and refine the results to the next finer scale sequentially. This will be done in future research.

For the zebra image, W_1 segments out the black and white stripes while W_2 and W_3 treat the zebra as a texture region. The synthesized images $\mathbf{I}_i^{\text{syn}} \sim p(\mathbf{I}|W_i), i = 1, 2, 3$ show that the texture model is not sufficient because we choose only 8 small filters for computational ease. Also the spline surface model plays an important role in segmenting the ground and background grass, and this is verified by the global shading changes in $\mathbf{I}_2^{\text{syn}}$ and $\mathbf{I}_3^{\text{syn}}$.

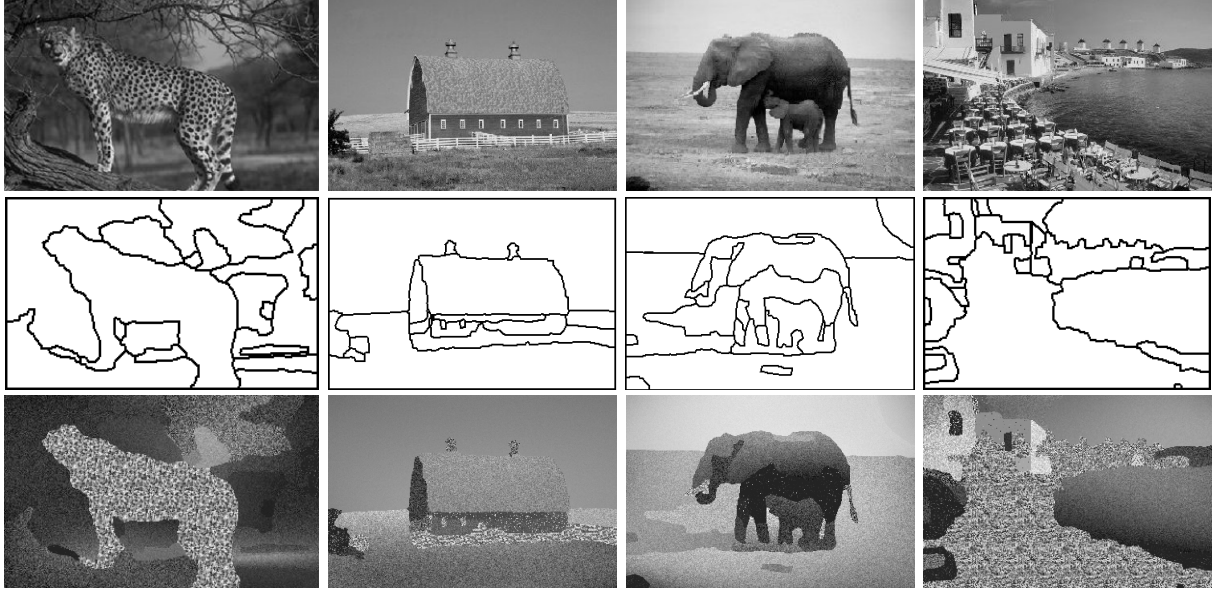


Figure 8.15: Grey level image segmentation by DDMCMC. Top: input images, middle: segmentation results W , bottom: synthesized images $\mathbf{I}^{\text{syn}} \sim p(\mathbf{I}|W)$ with the segmentation results W .

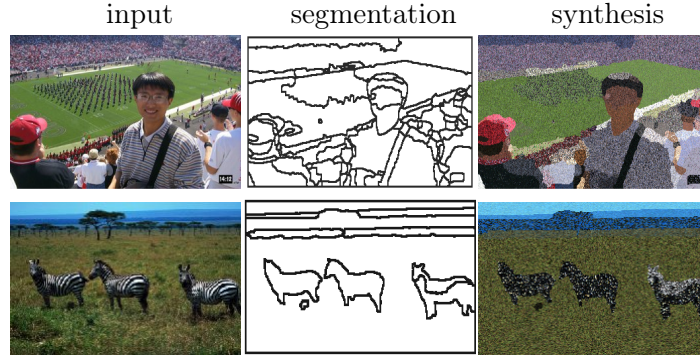


Figure 8.16: Color image segmentation by DDMCMC. Left: input images, middle: segmentation results W , right: synthesized images $\mathbf{I}^{\text{syn}} \sim p(\mathbf{I}|W)$ with the segmentation results W .

Figures 8.15 and 8.16 display some other grey level and color images using the same algorithm. We show the input (left) and a segmentation (middle) starting with arbitrary initial conditions and a synthesized image (right) drawn from the likelihood $\mathbf{I}^{\text{syn}} \sim p(\mathbf{I}|W)$. The γ values for these images are mostly set up as 1.5 with a few obtained at 1.0-3.5. It took about 10-30 minutes, depending upon the complexity of image contents, on a Pentium III PC to segment an image with medium size such as 350×250 pixels after learning the pseudo-likelihood texture models at the beginning.

The synthesis images show that we need to engage more stochastic models such as point, curve

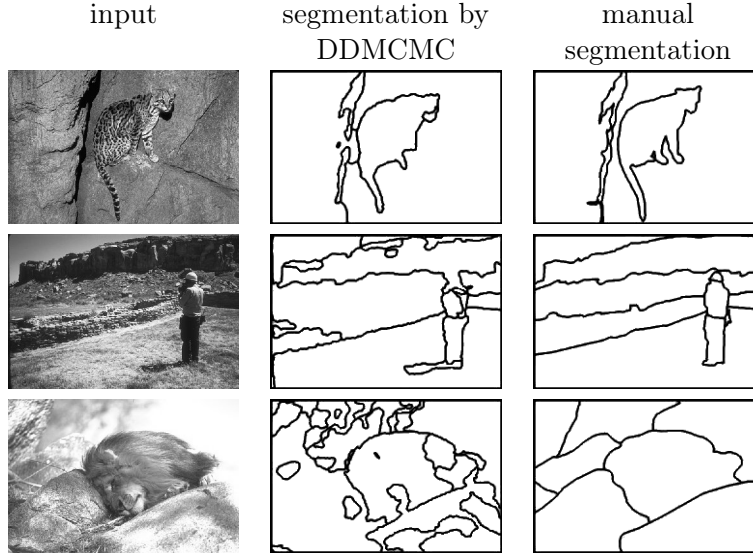


Figure 8.17: Some segmentation results by DDMCMC for the benchmark test by Martin. The errors for the above results by DDMCMC (middle) compared with the results by a human subject (right) are 0.1083, 0.3082, and 0.5290 respectively according to their metrics.

process, and object like faces etc. For example, in the first row of Figure 8.16. The music band in a football stadium forms a point process which is not captured. The face is also missing in the synthesis.

Figure 8.17 shows three grey level images out of the 50 natural images in both color and grey level for the benchmark study. The input (left), the segmentation results by DDMCMC (middle), and the manual segmentation by a human subject (right) are displayed.

8.8 Application: Image Parsing

We define image parsing to be the task of decomposing an image \mathbf{I} into its constituent visual patterns. The output is represented by a hierarchical graph W — called the “parsing graph”. The goal is to optimize the Bayesian posterior probability $p(W|\mathbf{I})$. Figure 8.18 illustrates a typical example where a football scene is first divided into three parts at a coarse level: a person in the foreground, a sports field, and the spectators. These three parts are further decomposed into nine visual patterns in the second level: a face, three texture regions, some text, a point process (the band on the field), a curve process (the markings on the field), a color region, and a region for nearby people. In principle, we can continue decomposing these parts until we reach a resolution criterion. The parsing graph is similar in spirit to the parsing trees used in speech and natural language processing [133] except that it can include horizontal connections (see the dashed curves in Figure 8.18) for specifying spatial relationships and boundary sharing between different visual patterns.

As in natural language processing, the parsing graph is not fixed and depends on the input image(s). An image parsing algorithm must *construct* the parsing graph on the fly⁷. Our image parsing algorithm consists of a set of reversible Markov chain jumps [87] with each type of jump corresponding to an operator for *reconfiguring* the parsing graph (i.e. creating or deleting nodes or changing the values of node attributes). These jumps combine to form an ergodic and reversible Markov chain in the space of possible parsing graphs. The Markov chain probability is guaranteed to converge to the invariant probability $p(W|\mathbf{I})$ and the Markov chain will simulate fair

⁷Unlike most graphical inference algorithms in the literature which assume fixed graphs, see belief propagation [212].

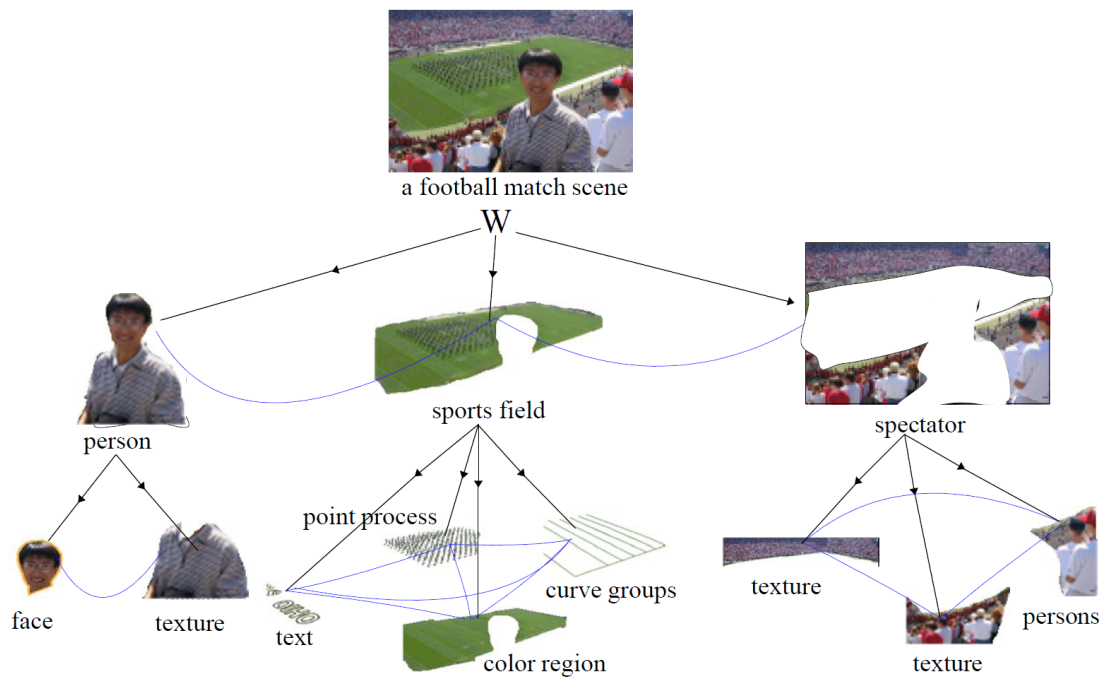


Figure 8.18: Image parsing example. The parsing graph is hierarchical and combines generative models (downward arrows) with horizontal connections (dashed lines), which specify spatial relationships between the visual patterns. See Figure 8.21 for a more abstract representation including variables for the node attributes.

samples from this probability.⁸ Our approach is built on previous work on Data-Driven Markov Chain Monte Carlo (DDMCMC) for recognition [225], segmentation [193], grouping [194] and graph partitioning [8, 9].

Image parsing seeks a full generative explanation of the input image in terms of generative models, $p(\mathbf{I}|W)$ and $p(W)$, for the diverse visual patterns which occur in natural images, see Figure 8.18. This differs from other computer vision tasks, such as segmentation, grouping, and recognition. These are usually performed by isolated vision modules which only seek to explain parts of the image. The image parsing approach enables these different modules to cooperate and compete to give a consistent interpretation of the entire image.

The integration of visual modules is of increasing importance as progress on the individual modules starts approaching performance ceilings. In particular, work on segmentation [65, 175, 193] and edge detection [22, 106] has reached performance levels where there seems little room for improvement when only low-level cues are used. For example, the segmentation failures in Figure 8.19 can only be resolved by combining segmentation with object detection and recognition. There has also recently been very successful work on the detection and recognition of objects [13, 126, 161, 199, 203, 206] and the classification of natural scenes [11, 147] using, broadly speaking, discriminative methods based on local bottom-up tests.

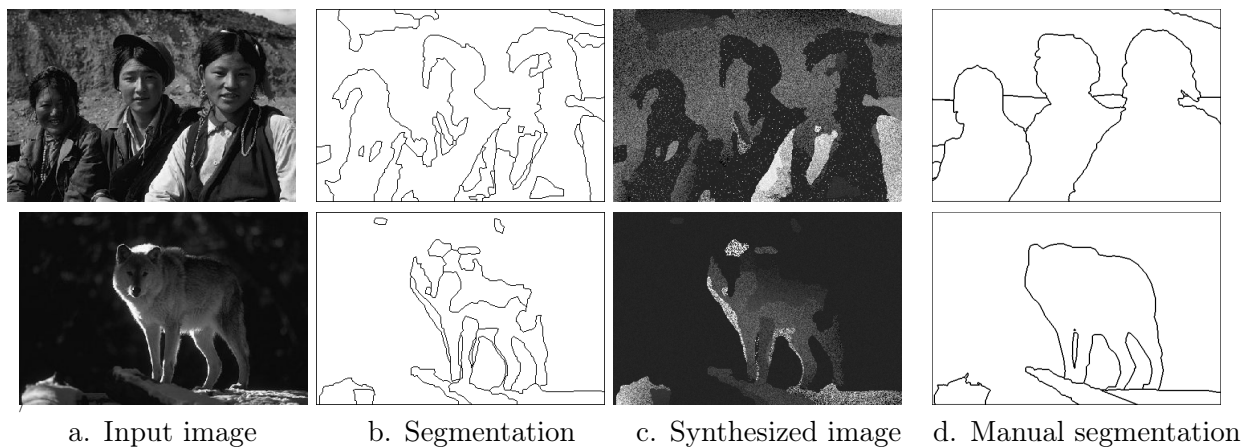


Figure 8.19: Examples of image segmentation failure by an algorithm [193] which uses only generic visual patterns (i.e. only low-level visual cues). The results (b) show that low-level visual cues are not sufficient to obtain good intuitive segmentations. The limitations of using only generic visual patterns are also clear in the synthesized images (c) which are obtained by stochastic sampling from the generative models after the parameters have been estimated by DDMCMC. The right panels (d) show the segmentations obtained by human subjects who, by contrast to the algorithm, appear to use object specific knowledge when doing the segmentation (though they were not instructed to) [136]. We conclude that to achieve good segmentation on these types of images requires combining segmentation with object detection and recognition.

But combining different visual modules requires a common framework which ensures consistency. Despite the effectiveness of discriminative methods for computing scene components, such as object labels and categories, they can also generate redundant and conflicting results. Mathematicians have argued [20] that discriminative methods must be followed by more sophisticated processes to (i) remove false alarms, (ii) amend missing objects by global context information, and (iii) reconcile

⁸For many natural images the posterior probabilities $P(W|\mathbf{I})$ are strongly peaked and so stochastic samples are close to the maxima of the posterior. So in this section we do not distinguish between sampling and inference (optimization).

conflicting (overlapping) explanations through model comparison. In this section, we impose such processes by using generative models for the entire image.

As we will show, our image parsing algorithm is able to integrate discriminative and generative methods so as to take advantage of their complementary strengths. Moreover, we can couple modules such as segmentation and object detection by our choice of the set of visual patterns used to parse the image. In this section, we focus on two types of patterns: – generic visual patterns for low/middle level vision, such as texture and shading, and object patterns at high level vision, such as frontal human faces and text.

These two types of patterns illustrate different ways in which the parsing graph can be constructed (see Figure 8.34 and the related discussion). The object patterns (face and text) have comparatively little variability so they can often be effectively detected as a whole by bottom-up tests and their parts can be located subsequently. Thus their parsing sub-graphs can be constructed in a “decompositional” manner from whole to parts. By contrast, a generic texture region has arbitrary shape and its intensity pattern has high entropy. Detecting such a region by bottom-up tests will require an enormous number of tests to deal with all this variability, and so will be computationally impractical. Instead, the parsing subgraphs should be built by grouping small elements in a “compositional” manner [18].

We illustrate our algorithm on natural images of complex city scenes and give examples where image segmentation can be improved by allowing object specific knowledge to disambiguate low-level cues, and conversely object detection can be improved by using generic visual patterns to explain away shadows and occlusions.

8.8.1 Bottom-Up and Top-Down Processing

A major element of our work is to integrate discriminative and generative methods for inference. In the recent computer vision literature, top-down and bottom-up procedures can be broadly categorized into two popular inference paradigms – *generative* methods for “top-down” and *discriminative* methods for “bottom-up”, illustrated in Figure 8.20. From this perspective, integrating generative and discriminative models is equivalent to combining bottom-up and top-down processing.

The role of bottom-up and top-down processing in vision has been often discussed. There is growing evidence (see [117, 182]) that humans can perform high level scene and object categorization tasks as fast as low level texture discrimination and other so-called pre-attentive vision tasks. This suggests that humans can detect both low and high level visual patterns at early stages in visual processing. It contrasts with traditional bottom-up feedforward architectures [135] which start with edge detection, followed by segmentation/grouping, before proceeding to object recognition and other high-level vision tasks. The experiments also relate to long standing conjectures about the role of the bottom-up/top-down loops in the visual cortical areas [145, 196], visual routines and pathways [195], the binding of visual cues [186], and neural network models such as the Helmholtz machine [49]. But although combining bottom-up and top-down processing is clearly important, there has not yet been a rigorous mathematical framework for how to achieve it.

In this section, we unify generative and discriminative approaches by designing an DDMCMC algorithm which uses discriminative methods to perform rapid inference of the parameters of generative models. From a computer vision perspective, DDMCMC combines bottom-up processing, implemented by the discriminative models, together with top-down processing by the generative models. The rest of this section gives an overview of our approach.

8.8.2 Generative and Discriminative Methods

Generative methods specify how the image \mathbf{I} is generated from the scene representation $W \in \Omega$. It combines a prior $p(W)$ and a likelihood function $p(\mathbf{I}|W)$ to give a joint posterior probability $p(W|\mathbf{I})$. These can be expressed as probability probabilities on graphs, where the input image \mathbf{I} is represented on the leaf nodes and W denotes the remaining nodes and node attributes of the graph. The structure of the graph, for example the number of nodes, is unknown and must be estimated for each input image.

Inference can be performed by stochastic sampling W from the posterior:

$$W \sim p(W|\mathbf{I}) \propto p(\mathbf{I}|W)p(W). \quad (8.28)$$

This enables us to estimate $W^* = \arg \max P(W|\mathbf{I})$.⁹ But the dimension of the sample space Ω is very high and so standard sampling techniques are computationally expensive.

By contrast, discriminative methods are very fast to compute. They do not specify models for how the image is generated. Instead they give discriminative (conditional) probabilities $q(w_j|\text{Tst}_j(\mathbf{I}))$ for components $\{w_j\}$ of W based on a sequence of bottom-up tests $\text{Tst}_j(\mathbf{I})$ performed on the image. The tests are based on local image features $\{F_{j,n}(\mathbf{I})\}$ which can be computed from the image in a cascade manner (e.g. AdaBoost filters, see Section (8.8.5)),

$$\text{Tst}_j(\mathbf{I}) = (F_{j,1}(\mathbf{I}), F_{j,2}(\mathbf{I}), \dots, F_{j,n}(\mathbf{I})), \quad j = 1, 2, \dots, K. \quad (8.29)$$

The following theorem shows that the KL-divergence between the true marginal posterior $p(w_j|\mathbf{I})$ and the optimal discriminant approximation $q(w_j|\text{Tst}(\mathbf{I}))$ using test $\text{Tst}(\mathbf{I})$ will decrease monotonically as new tests are added¹⁰.

Theorem 8.1. *The information gained for a variable w by a new test $\text{Tst}_+(\mathbf{I})$ is the decrease of Kullback-Leibler divergence between $p(w|\mathbf{I})$ and its best discriminative estimate $q(w|\text{Tst}_t(\mathbf{I}))$ or the increase of mutual information between w and the tests.*

$$\begin{aligned} & E_{\mathbf{I}}[KL(p(w|\mathbf{I}) \parallel q(w|\text{Tst}(\mathbf{I}))) - E_{\mathbf{I}}[KL(p(w|\mathbf{I}) \parallel q(w|\text{Tst}(\mathbf{I}), \text{Tst}_+(\mathbf{I})))] \\ &= MI(w \parallel \text{Tst}, \text{Tst}_+) - MI(w \parallel \text{Tst}) = E_{\text{Tst}, \text{Tst}_+} KL(q(w \mid \text{Tst}_t, \text{Tst}_+) \parallel q(w \mid \text{Tst}_t)) \geq 0, \end{aligned}$$

where $E_{\mathbf{I}}$ is the expectation with respect to $P(\mathbf{I})$, and $E_{\text{Tst}, \text{Tst}_+}$ is the expectation with respect to the probability on the test responses $(\text{Tst}, \text{Tst}_+)$ induced by $P(\mathbf{I})$.

The decrease of the Kullback-Leibler divergence equals zero if and only if $\text{Tst}(\mathbf{I})$ are sufficient statistics with respect to w .

In practice discriminative methods, particularly standard computer vision algorithms – see subsection (8.8.4), will typically only use a small number of features for computational practicality. Also their discriminative probabilities $q(w_j|\text{Tst}(\mathbf{I}))$ will often not be optimal. Fortunately the image parsing algorithm in this section only requires the discriminative probabilities $q(w_j|\text{Tst}(\mathbf{I}))$ to be rough approximations to $p(w_j|\mathbf{I})$.

The difference between discriminative and generative models is illustrated in Figure 8.20. Discriminative models are fast to compute and can be run in parallel because different components are computed independently (see arrows in Figure 8.20). But the components $\{w_i\}$ may not yield a consistent solution W and, moreover, W may not specify a consistent model for generating the observed image \mathbf{I} . These inconsistencies are indicated by the crosses in Figure 8.20. Generative models ensure consistency but require solving a difficult inference problem. It is an open problem whether discriminative models can be designed to infer the entire state W for the complicated generative models that we are dealing with. Mathematicians [20] have argued that this will not be practical and that discriminative models will always require additional post-processing.

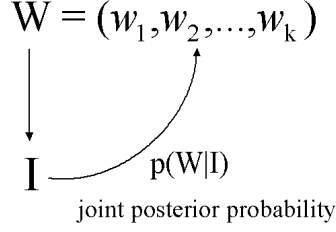
8.8.3 Markov Chain kernels and sub-kernels

Formally, our DDMCMC image parsing algorithm simulates a Markov chain $\mathcal{MC} = \langle \Omega, \nu, \mathcal{K} \rangle$ with kernel \mathcal{K} in space Ω and with probability ν for the starting state. An element $W \in \Omega$ is a parsing graph. We let the set of parsing graphs Ω be finite as images have finite pixels and grey levels.

⁹We are assuming that there are no known algorithms for estimating W^* directly.

¹⁰The optimal approximation occurs when $q(w_j|\text{Tst}(\mathbf{I}))$ equals the probability $p(w_j|\text{Tst}(\mathbf{I}))$ induced by $P(\mathbf{I}|W)P(W)$.

Generative methods



Discriminative methods

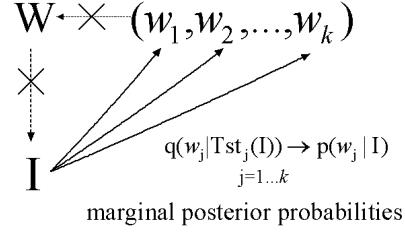


Figure 8.20: Comparison of two inference paradigms: Top-down *generative* methods versus bottom-up *discriminative* methods. The generative method specifies how the image \mathbf{I} can be synthesized from the scene representation W . By contrast, the discriminative methods are based by performing tests $Tst_j(\mathbf{I})$ and are not guaranteed to yield consistent solutions, see crosses explained in the text.

We proceed by defining a set of moves for reconfiguring the graph. These include moves to: (i) create nodes, (ii) delete nodes, and (iii) change node attributes. We specify stochastic dynamics for these moves in terms of transition kernels¹¹.

For each move we define a Markov Chain sub-kernel by a transition matrix $\mathcal{K}_a(W'|W : \mathbf{I})$ with $a \in \mathcal{A}$ being an index. This represents the probability that the system makes a transition from state W to state W' when sub-kernel a is applied (i.e. $\sum_{W'} \mathcal{K}_a(W'|W : \mathbf{I}) = 1, \forall W$). Kernels which alter the graph structure are grouped into reversible pairs. For example, the sub-kernel for node creation $\mathcal{K}_{a,r}(W'|W : \mathbf{I})$ is paired with the sub-kernel for node deletion $\mathcal{K}_{a,l}(W'|W : \mathbf{I})$. This can be combined into a paired sub-kernel $\mathcal{K}_a = \rho_{ar}\mathcal{K}_{a,r}(W'|W : \mathbf{I}) + \rho_{al}\mathcal{K}_{a,l}(W'|W : \mathbf{I})$ ($\rho_{ar} + \rho_{al} = 1$). This pairing ensures that $\mathcal{K}_a(W'|W : \mathbf{I}) = 0$ if, and only if, $\mathcal{K}_a(W|W' : \mathbf{I}) = 0$ for all states $W, W' \in \Omega$. The sub-kernels (after pairing) are constructed to obey the detailed balance condition:

$$p(W|\mathbf{I})\mathcal{K}_a(W'|W : \mathbf{I}) = p(W'|\mathbf{I})\mathcal{K}_a(W|W' : \mathbf{I}). \quad (8.30)$$

The full transition kernel is expressed as:

$$\mathcal{K}(W'|W : \mathbf{I}) = \sum_a \rho(a : \mathbf{I})\mathcal{K}_a(W'|W : \mathbf{I}), \quad \sum_a \rho(a : \mathbf{I}) = 1, \quad \rho(a : \mathbf{I}) > 0. \quad (8.31)$$

To implement this kernel, at each time step the algorithm selects the choice of move with probability $\rho(a : \mathbf{I})$ for move a , and then uses kernel $\mathcal{K}_a(W'|W; \mathbf{I})$ to select the transition from state W to state W' . Note that both probabilities $\rho(a : \mathbf{I})$ and $\mathcal{K}_a(W'|W; \mathbf{I})$ depend on the input image \mathbf{I} . This distinguishes our DDMMCMC methods from conventional MCMC computing [25, 123].

The full kernel obeys detailed balance, equation (8.30), because all the sub-kernels do. It will also be ergodic, provided the set of moves is sufficient (i.e. so that we can transition between any two states $W, W' \in \Omega$ using these moves). These two conditions ensure that $p(W|\mathbf{I})$ is the invariant (target) probability of the Markov Chain [25] in the finite space Ω .

Applying the kernel $\mathcal{K}_{a(t)}$ updates the Markov chain state probability $\mu_t(W)$ at step t to $\mu_{t+1}(W')$ at $t + 1$,¹²:

$$\mu_{t+1}(W') = \sum_W \mathcal{K}_{a(t)}(W'|W : \mathbf{I})\mu_t(W). \quad (8.32)$$

¹¹We choose stochastic dynamics because the Markov chain probability is guaranteed to converge to the posterior $P(W|\mathbf{I})$. The complexity of the problem means that deterministic algorithms for implementing these moves risk getting stuck in local minima.

¹²Algorithms like belief propagation [212] can be derived as approximations to this update equation by using a Gibbs sampler and making independence assumptions.

In summary, the DDMCMC image parser simulates a Markov chain \mathcal{MC} with a unique invariant probability $p(W|\mathbf{I})$. At time t , the Markov chain state (i.e. the parse graph) W follows a probability μ_t which is the product of the sub-kernels selected up to time t ,

$$W \sim \mu_t(W) = \nu(W_o) \cdot [\mathcal{K}_{a(1)} \circ \mathcal{K}_{a(2)} \circ \dots \circ \mathcal{K}_{a(t)}](W_o, W) \longrightarrow p(W|\mathbf{I}). \quad (8.33)$$

where $a(t)$ indexes the sub-kernel selected at time t . As the time t increases, $\mu_t(W)$ approaches the posterior $p(W|\mathbf{I})$ monotonically [25] at a geometric rate [54]. The following convergence theorem is useful for image parsing because it helps quantify the effectiveness of the different sub-kernels.

Theorem 8.2. *The Kullback-Leibler divergence between the posterior $p(W|\mathbf{I})$ and the Markov chain state probability decreases monotonically when a sub-kernel $\mathcal{K}_{a(t)}$, $\forall a(t) \in \mathcal{A}$ is applied,*

$$KL(p(W|\mathbf{I}) || \mu_t(W)) - KL(p(W|\mathbf{I}) || \mu_{t+1}(W)) \geq 0 \quad (8.34)$$

The decrease of KL-divergence is strictly positive and is equal to zero only after the Markov chain becomes stationary, i.e. $\mu = p$.

Proof. See [191].

The theorem is related to the second law of thermodynamics [44], and its proof makes use of the detailed balance equation (8.30). This KL divergence gives a measure of the “power” of each sub-kernel $\mathcal{K}_{a(t)}$ and so it suggests an efficient mechanism for selecting the sub-kernels at each time step. By contrast, classic convergence analysis shows that the convergence of the Markov Chain is exponentially fast, but does not give measures of power of sub-kernels.

8.8.4 DDMCMC and Proposal Probabilities

We now describe how to design the sub-kernels using proposal probabilities and discriminative models. This is at the heart of DDMCMC.

Each sub-kernel¹³ is designed to be of Metropolis-Hastings form [91, 139]:

$$\mathcal{K}_a(W'|W : \mathbf{I}) = Q_a(W'|W : \text{Tst}_a(\mathbf{I})) \min\left\{1, \frac{p(W'|\mathbf{I})Q_a(W|W' : \text{Tst}_a(\mathbf{I}))}{p(W|\mathbf{I})Q_a(W'|W : \text{Tst}_a(\mathbf{I}))}\right\}, \quad W' \neq W \quad (8.35)$$

where a transition from W to W' is proposed (stochastically) by the proposal probability $Q_a(W'|W : \text{Tst}_a(\mathbf{I}))$ and accepted (stochastically) by the acceptance probability:

$$\alpha(W'|W : \mathbf{I}) = \min\left\{1, \frac{p(W'|\mathbf{I})Q_a(W|W' : \text{Tst}_a(\mathbf{I}))}{p(W|\mathbf{I})Q_a(W'|W : \text{Tst}_a(\mathbf{I}))}\right\}. \quad (8.36)$$

Metropolis-Hastings ensures that the sub-kernels obey detailed balance (after pairing).

The proposal probability $Q_a(W'|W : \text{Tst}_a(\mathbf{I}))$ is a product (factorized) of some discriminative probabilities $q(w_j|\text{Tst}_j(\mathbf{I}))$ for the respective elements w_j changed in the move between W and W' (see later section). $\text{Tst}_a(\mathbf{I})$ is the set of bottom-up tests used in the proposal probabilities $Q_a(W'|W : \text{Tst}_a(\mathbf{I}))$ and $Q_a(W|W' : \text{Tst}_a(\mathbf{I}))$. The proposal probabilities must be fast to compute (because they should be evaluated for all the possible state W' that sub-kernel a can reach) and they should propose transitions to states W' where the posterior $p(W'|\mathbf{I})$ is likely to be high. The acceptance probabilities are more computationally expensive, because of their dependence on $p(W'|\mathbf{I})$, but they only need to be evaluated for the proposed state.

The design of the proposals is a trade-off. Ideally the proposals would be sampled from the posterior $p(W'|\mathbf{I})$, but this is impractical. Instead the trade-off requires: (i) the possibility of making large moves in Ω at each time step, (ii) the proposals should encourage moves to states with high posterior probability, and (iii) the proposals must be fast to compute.

¹³Except for one that evolves region boundaries.

More formally, we define the scope $\Omega_a(W) = \{W' \in \Omega : \mathcal{K}_a(W'|W : \mathbf{I}) > 0\}$ to be the set of states which can be reached from W in one time step using sub-kernel a . We want the scope $S_a(W)$ to be large so that we can make large moves in the space Ω at each time step (i.e. jump towards the solution and not crawl). The scope should also, if possible, include states W' with high posterior $p(W'|\mathbf{I})$ (i.e. it is not enough for the scope to be large, it should also be in the right part of Ω).

The proposals $Q_a(W'|W : \text{Tst}_a(\mathbf{I}))$ should be chosen so as to approximate

$$\frac{p(W'|\mathbf{I})}{\sum_{W'' \in \Omega_a(W)} p(W''|\mathbf{I})} \quad \text{if } W' \in \Omega_a(W), \quad = 0, \text{ otherwise.} \quad (8.37)$$

The proposals will be functions of the discriminative models for the components of W' and of the generative models for the current state W (because it is computationally cheap to evaluate the generative models for the current state). The details of the model $p(W|\mathbf{I})$ will determine the form of the proposals and how large we can make the scope while keeping the proposals easy to compute and able to approximate equation (8.37). See the detailed examples given in Section (8.8.5).

This description gives the bare bones of DDMCMC. We refer to [194] for a more sophisticated discussion of these issues from an MCMC perspective. In the discussion section, we describe strategies to improve DDMCMX. Preliminary theoretical results for the convergence of DDMCMC are encouraging for a special case (see Appendix C).

Finally, in Appendix D, we address the important practical issue of how to maintain detailed balance when there are multiple routes to transition between two state W and W' . We describe two ways to do this and the trade-offs involved.

Generative models and Bayesian formulation

This section describes the parsing graph and the generative models used for our image parsing algorithm in this section.

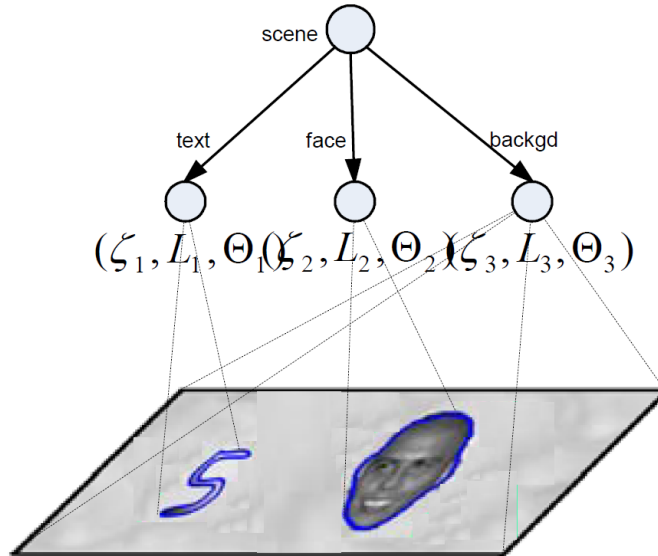


Figure 8.21: Abstract representation of the parsing graph used in this section. The intermediate nodes represent the visual patterns. Their child nodes correspond to the pixels in the image.

Figure 8.18 illustrates the general structure of a parsing graph. In this section, we take a simplified two-layer-graph illustrated in Figure 8.21, which is fully specified in a generative sense. The top node (“root”) of the graph represents the whole scene (with a label). It has K *intermediate nodes* for the visual patterns (face, text, texture, and shading). Each visual pattern has a number

of pixels at the bottom (“leaves”). In this graph no horizontal connections are considered between the visual patterns except that they share boundaries and form a partition of the image lattice.

The number K of intermediate nodes is a random variable, and each node $i = 1, \dots, K$ has a set of attributes (L_i, ζ_i, Θ_i) defined as follows. L_i is the shape descriptor and determines the region $R_i = R(L_i)$ of the image pixels covered by the visual pattern of the intermediate node. Conceptually, the pixels within R_i are child nodes of the intermediate node i . (Regions may contain holes, in which case the shape descriptor will have internal and external boundaries). The remaining attribute variables (ζ_i, Θ_i) specify the probability models $p(\mathbf{I}_{R(L_i)}|\zeta_i, L_i, \Theta_i)$ for generating the sub-image $\mathbf{I}_{R(L_i)}$ in region $R(L_i)$. The variables $\zeta_i \in \{1, \dots, 66\}$ indicate the visual pattern type (3 types of generic visual patterns, 1 face pattern, and 62 text character patterns), and Θ_i denotes the model parameters for the corresponding visual pattern (details are given in the following sections). The complete scene description can be summarized by:

$$W = (K, \{(\zeta_i, L_i, \Theta_i) : i = 1, 2, \dots, K\}).$$

The shape descriptors $\{L_i : i = 1, \dots, K\}$ are required to be consistent so that each pixel in the image is a child of one, and only one, of the intermediate nodes. The shape descriptors must provide a partition of the image lattice $\Lambda = \{(m, n) : 1 \leq m \leq \text{Height}(\mathbf{I}), 1 \leq n \leq \text{Width}(\mathbf{I})\}$ and hence satisfy the condition

$$\Lambda = \cup_{i=1}^K R(L_i), \quad R(L_i) \cap R(L_j) = \emptyset, \quad \forall i \neq j.$$

The generation process from the scene description W to \mathbf{I} is governed by the likelihood function:

$$p(\mathbf{I}|W) = \prod_{i=1}^K p(\mathbf{I}_{R(L_i)}|\zeta_i, L_i, \Theta_i).$$

The prior probability $p(W)$ is defined by

$$p(W) = p(K) \prod_{i=1}^K p(L_i)p(\zeta_i|L_i)p(\Theta_i|\zeta_i).$$

Under the Bayesian formulation, parsing the image corresponds to computing the W^* that maximizes *a posteriori* probability over Ω , the solution space of W ,

$$W^* = \arg \max_{W \in \Omega} p(W|\mathbf{I}) = \arg \max_{W \in \Omega} p(\mathbf{I}|W)p(W). \quad (8.38)$$

It remains to specify the prior $p(W)$ and the likelihood function $p(\mathbf{I}|W)$. We set the prior terms $p(K)$ and $p(\Theta_i|\zeta_i)$ to be uniform probabilities. The term $p(\zeta_i|L_i)$ is used to penalize high model complexity and was estimated for the three generic visual patterns from training data in [193].

Shape models

We use two types of shape descriptor in this section. The first is used to define shapes of generic visual patterns and faces. The second defines the shapes of text characters.

1. Shape descriptors for generic visual patterns and faces

In this case, the shape descriptor represents the boundary¹⁴ of the image region by a list of pixels $L_i = \partial R_i$. The prior is defined by:

$$p(L_i) \propto \exp\{-\gamma|R(L_i)|^\alpha - \lambda|L_i|\}. \quad (8.39)$$

In this section, we set $\alpha = 0.9$. For computational reasons, we use this prior for face shapes though more complicated priors [42] can be applied.

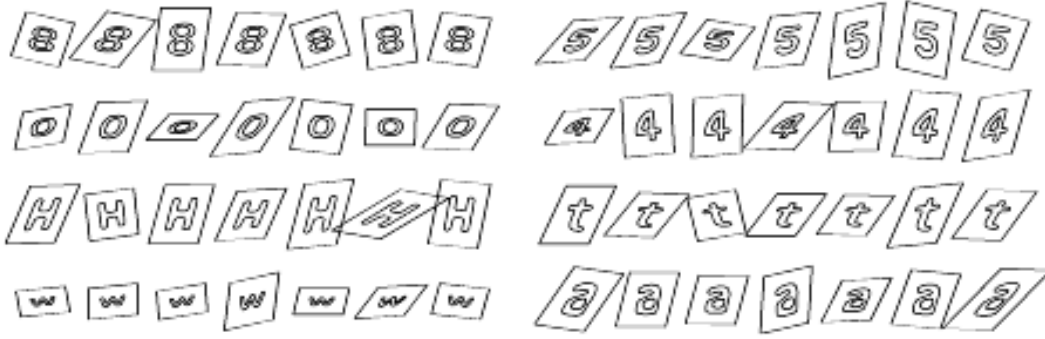


Figure 8.22: Random samples drawn from the shape descriptors for text characters.

2. Shape descriptors for text characters

We model text characters by 62 deformable templates corresponding to the ten digits and the twenty six letters in both upper and lower cases. These deformable templates are defined by 62 prototype characters and a set of deformations. The prototypes are represented by an outer boundary and, at most, two inner boundaries. Each boundary is modeled by a B-spline using twenty five control points. The prototype characters are indexed by $c_i \in \{1, \dots, 62\}$ and their control points are represented by a matrix $TP(c_i)$.

We now define two types of deformations on the templates. One is a global affine transformation, and the other is a local elastic deformation. First we allow the letters to be deformed by an affine transform M_i . We put a prior $p(M_i)$ to penalize severe rotation and distortion. This is obtained by decomposing M_i as:

$$M_i = \begin{pmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & h \\ 0 & 1 \end{pmatrix}.$$

where θ is the rotation angle, σ_x and σ_y denote scaling, and h is for shearing. The prior on M_i is

$$p(M_i) \propto \exp\{-a|\theta|^2 + b(\frac{\sigma_x}{\sigma_y} + \frac{\sigma_y}{\sigma_x})^2 + ch^2\},$$

where a, b, c are parameters.

Next, we allow local deformations by adjusting the positions of the B-spline control points. For a digit/letter c_i and affine transform M_i , the contour points of the template are given by $G_{TP}(M_i, c_i) = U \times M_s \times M_i \times TP(c_i)$. Similarly the contour points on the shape with control points S_i are given by $G_S(M_i, c_i) = U \times M_s \times S_i$ (U and M_s are the B-Spline matrices). We define a probability distribution $p(S_i|M_i, c_i)$ for the elastic deformation given by S_i ,

$$p(S_i|M_i, c_i) \propto \exp\{-\gamma|R(L_i)|^\alpha - D(G_S(M_i, c_i)||G_{TP}(M_i, c_i))\},$$

where $D(G_S(M_i, c_i)||G_{TP}(M_i, c_i))$ is the overall distance between contour template and the deformed contour (these deformations are small so the correspondence between points on the curves can be obtained by nearest neighbor matches, see [192] for how we can refine this). Figure 8.22 shows some samples drawn from the above model.

In summary, each deformable template is indexed by $c_i \in \{1..62\}$ and has a shape descriptor:

$$L_i = (c_i, M_i, S_i),$$

The prior distribution on L_i is specified by:

$$p(L_i) = p(c_i)p(M_i)p(S_i|M_i, c_i).$$

¹⁴The boundary can include an “internal boundary” if there is a hole inside the image region explained by a different visual pattern.

Here $p(c_i)$ is a uniform distribution on all the digits and letters (we do not place a prior distribution on text strings, though it is possible to do so [102]).

Generative intensity models

We use four families of generative intensity models for describing intensity patterns of (approximately) constant intensity, clutter/texture, shading, and face. The first three are similar as those defined in [193].

1. Constant intensity model $\zeta = 1$:

This assumes that pixel intensities in a region R are subject to independently and identically distributed (iid) Gaussian distribution,

$$p_1(\mathbf{I}_{R(L)}|\zeta = 1, L, \Theta) = \prod_{v \in R(L)} G(\mathbf{I}_v - \mu; \sigma^2), \quad \Theta = (\mu, \sigma)$$

2. Clutter/texture model $\zeta = 2$:

This is a non-parametric intensity histogram $h()$ discretized to take G values (i.e. is expressed as a vector (h_1, h_2, \dots, h_G)). Let n_j be the number of pixels in $R(L)$ with intensity value j .

$$p_2(\mathbf{I}_{R(L)}|\zeta = 2, L, \Theta) = \prod_{v \in R(L)} h(\mathbf{I}_v) = \prod_{j=1}^G h_j^{n_j}, \quad \Theta = (h_1, h_2, \dots, h_G).$$

3. Shading model $\zeta = 3$ and $\zeta = 5, \dots, 66$:

This family of models are used to describe generic shading patterns, and text characters. We use a quadratic form

$$J(x, y; \Theta) = ax^2 + bxy + cy^2 + dx + ey + f,$$

with parameters $\Theta = (a, b, c, d, e, f, \sigma)$. Therefore, the generative model for pixel (x, y) is

$$p_3(\mathbf{I}_{R(L)}|\zeta \in \{3, (5, \dots, 66)\}, L, \Theta) = \prod_{v \in R(L)} G(\mathbf{I}_v - J_v; \sigma^2), \quad \Theta = (a, b, c, d, e, f, \sigma).$$

4. The PCA face model $\zeta = 4$:



Figure 8.23: Random samples drawn from the PCA face model.

The generative model for faces is simpler and uses Principal Component Analysis (PCA) to obtain representations of the faces in terms of principal components $\{B_i\}$ and covariances Σ . Lower level features, also modeled by PCA, can be added [143]. Figure 8.23 shows some faces sampled from the PCA model. We also add other features such as the occlusion process, as described in Hallinan et al [89].

$$p_4(\mathbf{I}_R(L)|\zeta = 4, L, \Theta) = G(\mathbf{I}_R(L) - \sum_i \lambda_i B_i; \Sigma), \quad \Theta = (\lambda_1, \dots, \lambda_n, \Sigma).$$

Overview of the Algorithm

This section gives the control structure of an image parsing algorithm, and Figure 8.25 shows the algorithm's diagram. Our algorithm must construct the parse graph on the fly and to estimate the scene interpretation W .

Figure 8.24 illustrates how the algorithm selects the Markov chain moves (dynamics or sub-kernels) to search through the space of possible parse graphs of the image by altering the graph structure (by deleting or adding nodes) and by changing the node attributes. An equivalent way of visualizing the algorithm is in terms of a search through the solution space Ω , see [193, 194] for more details of this viewpoint.

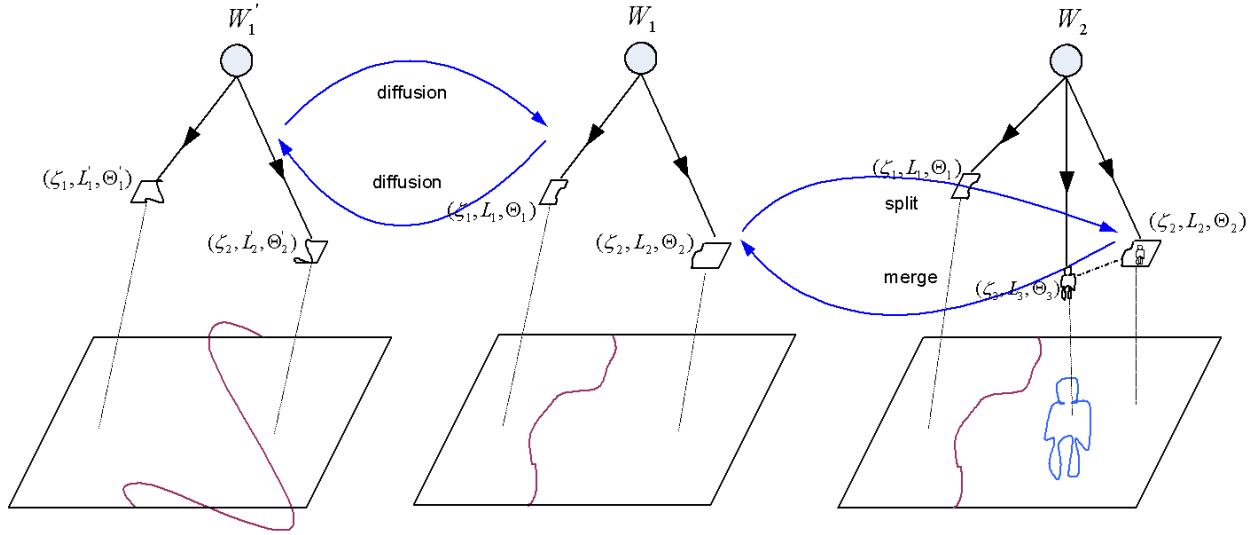


Figure 8.24: Examples of Markov chain dynamics that change the graph structure or the node attributes of the graph giving rise to different ways to parse the image.

We first define the set of moves to reconfigure the graph. These are: (i) birth or death of face nodes, (ii) birth or death of text characters, (iii) splitting or merging of regions, (iv) switching node attributes (region type ζ_i and model parameters Θ_i), (v) boundary evolution (altering the shape descriptors L_i of nodes with adjacent regions). These moves are implemented by sub-kernels. The first four moves are reversible jumps [87], and will be implemented by the Metropolis-Hastings equation (8.35). The fifth move, boundary evolution, is implemented by a stochastic partial differential equation.

The sub-kernels for these moves require proposal probabilities driven by elementary discriminative methods, which we review in the next subsection. The proposal probabilities are designed using the criteria in subsection (8.8.4), and full details are given in Section (8.8.5).

The control structure of the algorithm is described in Section (8.8.4). The full transition kernel for the image parser is built by combining the sub-kernels, as described in subsection (8.8.3) and

Figure 8.25. The algorithm proceeds (stochastically) by selecting a sub-kernel, selecting where in the graph to apply it, and then deciding whether or not to accept the operation.

Discriminative Methods

The discriminative methods give approximate posterior probabilities $q(w_j|\text{Tst}_j(\mathbf{I}))$ for the elementary components w_j of W . For computational efficiency, these probabilities are based only on a small number of simple tests $\text{Tst}_j(\mathbf{I})$.

We briefly overview and classify the discriminative methods used in our implementation. Section (8.8.5) shows how these discriminative methods are composed, see crosses in Figure 8.25, to give proposals for making moves in the parsing graph.

1. Edge Cues. These cues are based on edge detectors [29], [22], [106]. They are used to give proposals for region boundaries (i.e. the shape descriptor attributes of the nodes). Specifically, we run the Canny detector at three scales followed by edge linking to give partitions of the image lattice. This gives a finite list of candidate partitions which are assigned weights, see section (8.8.5) and [193]. The discriminative probability is represented by this weighted list of particles. In principle, statistical edge detectors [106] would be preferable to Canny because they give discriminative probabilities $q(w_j|\text{Tst}_j(\mathbf{I}))$ learnt from training data.

2. Binarization Cues. These cues are computed using a variant of Niblack’s algorithm [151]. They are used to propose boundaries for text characters (i.e. shape descriptors for text nodes), and will be used in conjunction with proposals for text detection. The binarization algorithm, and an example of its output, are given in Section (8.8.5). Like edge cues, the algorithm is run at different parameters settings and represents the discriminative probability by a weighted list of particles indicating candidate boundary locations.

3. Face Region Cues. These cues are learnt by a variant of AdaBoost [173], [199] which outputs discriminative probabilities [68], see Section (8.8.5). They propose the presence of faces in sub-regions of the image. These cues are combined with edge detection to propose the localization of faces in an image.

4. Text Region Cues. These cues are also learnt by a probabilistic version of AdaBoost, see Section (8.8.5). The algorithm is applied to image windows (at a range of scales). It outputs a discriminative probability for the presence of text in each window. Text region cues are combined with binarization to propose boundaries for text characters.

5. Shape Affinity Cues. These act on shape boundaries, produced by binarization, to propose text characters. They use shape context cues [13] and information features [192] to propose matches between the shape boundaries and the deformable template models of text characters.

6. Region Affinity Cues. These are used to estimate whether two regions R_i, R_j are likely to have been generated by the same visual pattern family and model parameters. They use an affinity similarity measure [175] of the intensity properties $\mathbf{I}_{R_i}, \mathbf{I}_{R_j}$.

7. Model Parameter and Visual Pattern Family cues. These are used to propose model parameters and visual pattern family identity. They are based on clustering algorithms, such as mean-shift [40]. The clustering algorithms depend on the model types and are described in [193].

In our current implementation, we conduct all the bottom-up tests $\text{Tst}_j(\mathbf{I}), j = 1, 2, \dots, K$ at an early stage for all the discriminative models $q_j(w_j|\text{Tst}_j(\mathbf{I}))$, and they are then combined to form composite tests $\text{Tst}_a(\mathbf{I})$ for each subkernel \mathcal{K}_a in equations (8.35,8.36).

Control Structure of the Algorithm

The control strategy used by our image parser is illustrated in Figure 8.25. The image parser explores the space of parsing graphs by a Markov Chain Monte Carlo sampling algorithm. This algorithm uses a transition kernel \mathcal{K} which is composed of sub-kernels \mathcal{K}_a corresponding to different ways to reconfigure the parsing graph. These sub-kernels come in reversible pairs¹⁵ (e.g. birth and death) and are designed so that the target probability distribution of the kernel is the generative posterior $p(W|\mathbf{I})$. At each time step, a sub-kernel is selected stochastically. The sub-kernels use the

¹⁵Except for the boundary evolution sub-kernel which will be described separately, see Section 8.8.5.

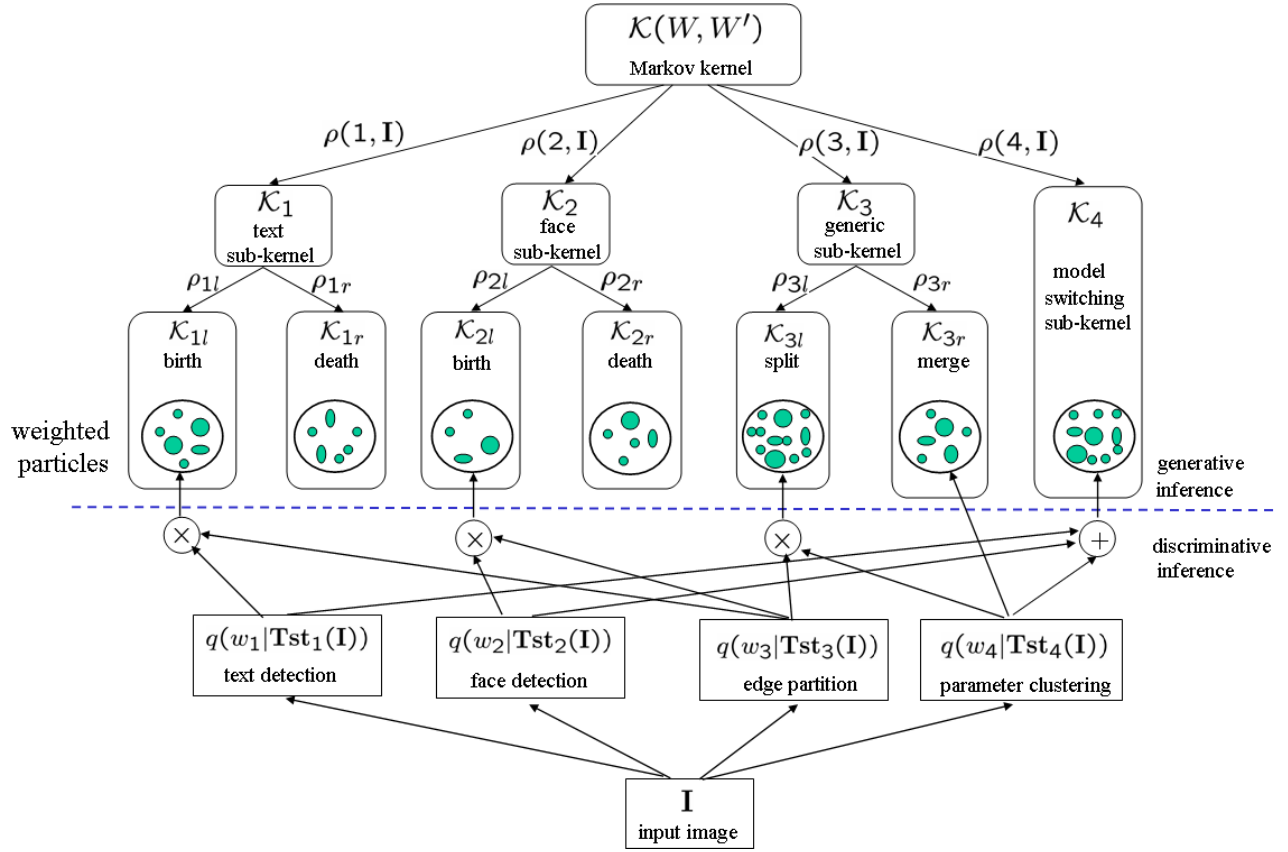


Figure 8.25: Integrating generative (top-down) and discriminative (bottom-up) methods for image parsing. This diagram illustrates the main points of the image parser. The dynamics are implemented by an ergodic Markov chain \mathcal{K} , whose invariant probability is the posterior $p(W|\mathbf{I})$, and which is composed of reversible sub-kernels \mathcal{K}_a for making different types of moves in the parse graph (e.g. giving birth to new nodes or merging nodes). At each time step the algorithm selects a sub-kernel stochastically. The selected sub-kernel proposes a specific move (e.g. to create or delete specific nodes) and this move is then evaluated and accepted stochastically, see equation (8.35). The proposals are based on both bottom-up (discriminative) and top-down (generative) processes, see subsection (8.8.4). The bottom-up processes compute discriminative probabilities $q(w_j|\text{Tst}_j(\mathbf{I}))$, $j = 1, 2, 3, 4$ from the input image \mathbf{I} based on feature tests $\text{Tst}_j(\mathbf{I})$. An additional sub-kernel for boundary evolution uses a stochastic partial differential equation will be described later.

Metropolis-Hasting sampling algorithm, see equation (8.35), which proceeds in two stages. First, it proposes a reconfiguration of the graph by sampling from a proposal probability. Then it accepts or rejects this reconfiguration by sampling the acceptance probability.

To summarize, we outline the algorithm below. At each time step, it specifies (stochastically) which move to select (i.e. which sub-kernel), where to apply it in the graph, and whether to accept the move. The probability to select moves $\rho(a : \mathbf{I})$ was first set to be independent of \mathbf{I} , but we got better performance by adapting it using discriminative cues to estimate the number of faces and text characters in the image (see details below). The choice of where to apply the move is specified (stochastically) by the sub-kernel. For some sub-kernels it is selected randomly and for others is chosen based on a *fitness factor* (see details in section (8.8.5)), which measures how well the current

model fits the image data. Some annealing is required to start the algorithm because of the limited scope of the moves in the current implementation (the need for annealing will be reduced if the compositional techniques described in [9]) are used).

We improved the effectiveness of the algorithm by making the move selection adapt to the image (i.e. by making $\rho(a : \mathbf{I})$ depend on \mathbf{I}). In particular, we increased the probability of giving birth and death of faces and text, $\rho(1)$ and $\rho(2)$, if the bottom-up (AdaBoost) proposals suggested that there are many objects in the scene. For example, let $N(\mathbf{I})$ be the number of proposals for faces or text above a threshold T_a . Then we modify the probabilities in the table by $\rho(a_1) \mapsto \{\rho(a_1) + kg(N(\mathbf{I}))\}/Z$, $\rho(a_2) \mapsto \{\rho(a_2) + kg(N)\}/Z$, $\rho(a_3) \mapsto \rho(a_3)/Z$, $\rho(a_4) \mapsto \rho(a_4)/Z$, where $g(x) = x$, $x \leq T_b$, $g(x) = T_b$, $x \geq T_b$ and $Z = 1 + 2k$ is chosen to normalize the probability.

The basic control strategy of the image parsing algorithm

1. Initialize W (e.g. by dividing the image into four regions), setting their shape descriptors, and assigning the remaining node attributes at random.
2. Set the temperature to be T_{init} .
3. Select the type a of move by sampling from a probability $\rho(a)$, with $\rho(1) = 0.2$ for faces, $\rho(2) = 0.2$ for text, $\rho(3) = 0.4$ for splitting and merging, $\rho(4) = 0.15$ for switching region model (type or model parameters), and $\rho(5) = 0.05$ for boundary evolution. This was modified slightly adaptively, see caption and text.
4. If the selected move is boundary evolution, then select adjacent regions (nodes) at random and apply stochastic steepest descent, see section (8.8.5).
5. If the jump moves are selected, then a new solution W' is randomly sampled as follows:
 - For the birth or death of a face, see section (8.8.5), we propose to create or delete a face. This includes a proposal for where in the image to do this.
 - For the birth or death of text, see section (8.8.5), we propose to create a text character or delete an existing one. This includes a proposal for where to do this.
 - For region splitting, see section (8.8.5), a region (node) is randomly chosen biased by its fitness factor. There are proposals for where to split it and for the attributes of the resulting two nodes.
 - For region merging, see section (8.8.5), two neighboring regions (nodes) are selected based on a proposal probability. There are proposals for the attributes of the resulting node.
 - For switching, see section (8.8.5), a region is selected randomly according to its fitness factor and a new region type and/or model parameters is proposed.
 - The full proposal probabilities, $Q(W|W : \mathbf{I})$ and $Q(W'|W : \mathbf{I})$ are computed.
 - The Metropolis-Hastings algorithm, equation (8.35), is applied to accept or reject the proposed move.
6. Reduce the temperature $T = 1 + T_{init} \times \exp(-t \times c|R|)$, where t is the current iteration step, c is a constant and $|R|$ is the size of the image.
7. Repeat the above steps and until the convergence criterion is satisfied (by reaching the maximum number of allowed steps or by lack of decrease of the negative log posterior).

8.8.5 The Markov Chain Kernels

Boundary Evolution

These moves evolve the positions of the region boundaries but preserve the graph structure. They are implemented by a stochastic partial differential equation (Langevin equation) driven by Brownian

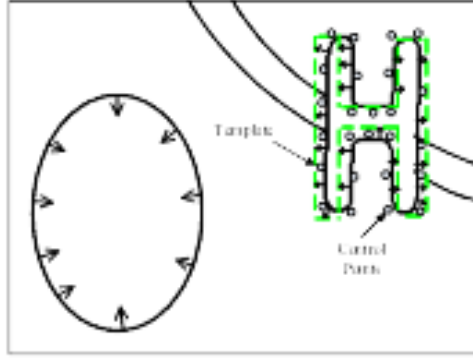


Figure 8.26: The evolution of the region boundaries is implemented by stochastic partial differential equations which are driven by models competing for ownership of the regions.

noise and can be derived from a Markov Chain [77]. The deterministic component of the PDE is obtained by performing steepest descent on the negative log-posterior, as derived in [224].

We illustrate the approach by deriving the deterministic component of the PDE for the evolution of the boundary between a letter T_j and a generic visual pattern region \mathbf{R}_i . The boundary will be expressed in terms of the control points $\{S_m\}$ of the shape descriptor of the letter. Let v denote a point on the boundary, i.e. $v(s) = (x(s), y(s))$ on $\Gamma(s) = \partial R_i \cap \partial R_j$. The deterministic part of the evolution equation is obtained by taking the derivative of the negative log-posterior $-\log p(W|\mathbf{I})$ with respect to the control points.

The relevant parts of the negative log-posterior, see equation (8.38,8.8.4) are given by $E(\mathbf{R}_i)$ and $E(T_j)$ where:

$$E(\mathbf{R}_i) = \int \int_{R_i} -\log p(\mathbf{I}(x, y) | \theta_{\zeta_i}) dx dy + \gamma |R_i|^\alpha + \lambda |\partial R_i|.$$

and

$$E(T_j) = \int \int_{L_j} \log p(\mathbf{I}(x, y) | \theta_{\zeta_j}) dx dy + \gamma |R(L_j)|^\alpha - \log p(L_j).$$

Differentiating $E(R_i) + E(T_j)$ with respect to the control points $\{S_m\}$ yields the evolution PDE:

$$\begin{aligned} \frac{dS_m}{dt} &= -\frac{\delta E(\mathbf{R}_i)}{\delta S_m} - \frac{\delta E(T_j)}{\delta S_m} \\ &= \int \left[-\frac{\delta E(\mathbf{R}_i)}{\delta v} - \frac{\delta E(T_j)}{\delta v} \right] \frac{1}{|\mathbf{J}(s)|} ds \\ &= \int \mathbf{n}(v) \left[\log \frac{p(\mathbf{I}(v); \theta_{\zeta_i})}{p(\mathbf{I}(v); \theta_{\zeta_j})} + \alpha \gamma \left(\frac{1}{|D_j|^{1-\alpha}} - \frac{1}{|D_i|^{1-\alpha}} \right) - \lambda \kappa + D(G_{S_j}(s) || G_T(s)) \right] \frac{1}{|\mathbf{J}(s)|} ds, \end{aligned}$$

where $\mathbf{J}(s)$ is the Jacobian matrix for the spline function. (Recall that $\alpha = 0.9$ in the implementation).

The log-likelihood ratio term $\log \frac{p(\mathbf{I}(v); \theta_{\zeta_i})}{p(\mathbf{I}(v); \theta_{\zeta_j})}$ implements the competition between the letter and the generic region models for ownership of the boundary pixels.

Markov chain sub-kernels

Changes in the graph structure are realized by Markov chain *jumps* implemented by four different sub-kernels.

Sub-kernel I: birth and death of text

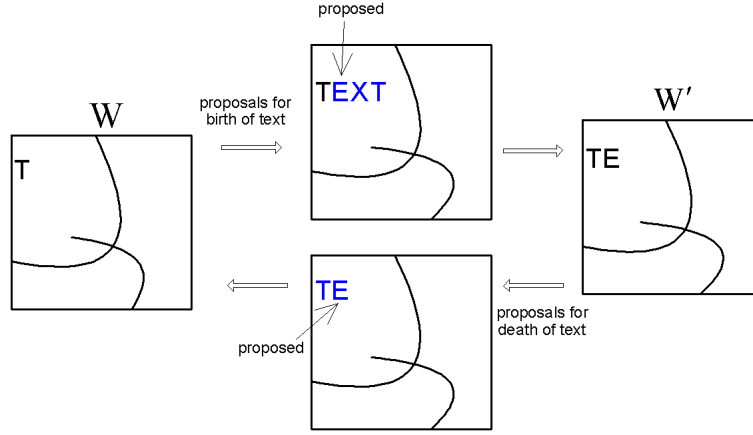


Figure 8.27: An example of the birth-death of text. State W consists of three generic regions and a character “T”. Proposals are computed for 3 candidate characters, “E”, “X”, and “T”, obtained by AdaBoost and binarization methods (see section (8.8.5)). One is selected, see arrow, which changes the state to W' . Conversely, there are 2 candidate in state W' and the one selected, see arrow, returns the system to state W .

This pair of jumps is used to create or delete text characters. We start with a parse graph W and transition into parse graph W' by creating a character. Conversely, we transition from W' back to W by deleting a character.

The proposals for creating and deleting text characters are designed to approximate the terms in equation (8.37). We obtain a list of candidate text character shapes by using AdaBoost to detect text regions followed by binarization to detect candidate text character boundaries within text regions (see section (8.8.5)). This list is represented by a set of particles which are weighted by the similarity to the deformable templates for text characters (see below):

$$S_{1r}(W) = \{ (z_{1r}^{(\mu)}, \omega_{1r}^{(\mu)}) : \mu = 1, 2, \dots, N_{1r} \}.$$

Similarly, we specify another set of weighted particles for removing text characters:

$$S_{1l}(W') = \{ (z_{1l}^{(\nu)}, \omega_{1l}^{(\nu)}) : \nu = 1, 2, \dots, N_{1l} \}.$$

$\{z_{1r}^{(\mu)}\}$ and $\{z_{1l}^{(\nu)}\}$ represent the possible (discretized) shape positions and text character deformable templates for creating or removing text, and $\{\omega_{1r}^{(\mu)}\}$ and $\{\omega_{1l}^{(\nu)}\}$ are their corresponding weights. The particles are then used to compute proposal probabilities

$$\mathbf{Q}_{1r}(\mathbf{W}'|\mathbf{W} : \mathbf{I}) = \frac{\omega_{1r}(\mathbf{W}')}{\sum_{\mu=1}^{N_{1r}} \omega_{1r}^{(\mu)}}, \quad \mathbf{Q}_{1l}(\mathbf{W}|\mathbf{W}', \mathbf{I}) = \frac{\omega_{1l}(\mathbf{W})}{\sum_{\nu=1}^{N_{1l}} \omega_{1l}^{(\nu)}}.$$

The weights $\omega_{1r}^{(\mu)}$ and $\omega_{1l}^{(\nu)}$ for creating new text characters are specified by shape affinity measures, such as shape contexts [13] and informative features [192]. For deleting text characters we calculate $\omega_{1l}^{(\nu)}$ directly from the likelihood and prior on the text character. Ideally these weights will approximate the ratios $\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})}$ and $\frac{p(W|\mathbf{I})}{p(W'|\mathbf{I})}$.

Sub-kernel II: birth and death of face

The sub-kernel for the birth and death of faces is very similar to the sub-kernel of birth and death of text. We use AdaBoost method discussed in sect. (8.8.5) to detect candidate faces. Face boundaries are obtained directly from using edge detection to give candidate face boundaries. The proposal probabilities are computed similarly to those for sub-kernel I.

Sub-kernel III: splitting and merging regions

This pair of jumps is used to create or delete nodes by splitting and merging regions (nodes). We start with a parse graph W and transition into parse graph W' by splitting node i into nodes j and k . Conversely, we transition back to W by merging nodes j and k into node i . The selection of which region i to split is based on a robust function on $p(\mathbf{I}_{R_i}|\zeta_i, L_i, \Theta_i)$ (i.e. the worse the model for region R_i fits the data, the more likely we are to split it). For merging, we use a region affinity measure [175] and propose merges between regions which have high affinity.

Formally, we define W, W' :

$$W = (K, (\zeta_k, L_k, \Theta_k), W_-) \Rightarrow W' = (K+1, (\zeta_i, L_i, \Theta_i), (\zeta_j, L_j, \Theta_j), W_-)$$

where W_- denotes the attributes of the remaining $K-1$ nodes in the graph.

We obtain proposals by seeking approximations to equation (8.37) as follows.

We first obtain three edge maps. These are given by Canny edge detectors [29] at different scales (see [193] for details). We use these edge maps to create a list of particles for splitting $S_{3r}(W)$. A list of particles for merging is denoted by $S_{3l}(W')$.

$$S_{3r}(W) = \{ (z_{3r}^{(\mu)}, \omega_{3r}^{(\mu)}) : \mu = 1, 2, \dots, N_{3r} \}, \quad S_{3l}(W') = \{ (z_{3l}^{(\nu)}, \omega_{3l}^{(\nu)}) : \nu = 1, 2, \dots, N_{3l} \}$$

where $\{z_{3r}^{(\mu)}\}$ and $\{z_{3l}^{(\nu)}\}$ represent the possible (discretized) positions for splitting and merging, and their weights $\{\omega_{3r}^{(\mu)}\}, \{\omega_{3l}^{(\nu)}\}$ will be defined shortly. In other words, we can *only* split a region i into regions j and k along a contour $z_{3r}^{(\mu)}$ (i.e. $z_{3r}^{(\mu)}$ forms the new boundary). Similarly we can only merge regions j and k into region i by deleting a boundary contour $z_{3l}^{(\nu)}$.

We now define the weights $\{\omega_{3r}^{(\mu)}\}, \{\omega_{3l}^{(\nu)}\}$. These weights will be used to determine probabilities for the splits and merges by:

$$\mathbf{Q}_{3r}(\mathbf{W}'|\mathbf{W}:\mathbf{I}) = \frac{\omega_{3r}(\mathbf{W}')}{\sum_{\mu=1}^{N_{3r}} \omega_{3r}^{(\mu)}}, \quad \mathbf{Q}_{3l}(\mathbf{W}|\mathbf{W}':\mathbf{I}) = \frac{\omega_{3l}(\mathbf{W})}{\sum_{\nu=1}^{N_{3l}} \omega_{3l}^{(\nu)}}.$$

Again, we would like $\omega_{3r}^{(\mu)}$ and $\omega_{3l}^{(\nu)}$ to approximate the ratios $\frac{p(W|\mathbf{I})}{p(W'|\mathbf{I})}$ and $\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})}$ respectively. $\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})}$ is given by:

$$\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})} = \frac{p(\mathbf{I}_{R_i}|\zeta_i, L_i, \Theta_i)p(\mathbf{I}_{R_j}|\zeta_j, L_j, \Theta_j)}{p(\mathbf{I}_{R_k}|\zeta_k, L_k, \Theta_k)} \cdot \frac{p(\zeta_i, L_i, \Theta_i)p(\zeta_j, L_j, \Theta_j)}{p(\zeta_k, L_k, \Theta_k)} \cdot \frac{p(K+1)}{p(K)}$$

This is expensive to compute, so we approximate $\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})}$ and $\frac{p(W|\mathbf{I})}{p(W'|\mathbf{I})}$ by:

$$\omega_{3r}^{(\mu)} = \frac{q(R_i, R_j)}{p(\mathbf{I}_{R_k}|\zeta_k, L_k, \Theta_k)} \cdot \frac{[q(L_i)q(\zeta_i, \Theta_i)][q(L_j)q(\zeta_j, \Theta_j)]}{p(\zeta_k, L_k, \Theta_k)}. \quad (8.40)$$

$$\omega_{3l}^{(\nu)} = \frac{q(R_i, R_j)}{p(\mathbf{I}_{R_i}|\zeta_i, L_i, \Theta_i)p(\mathbf{I}_{R_j}|\zeta_j, L_j, \Theta_j)} \cdot \frac{q(L_k)q(\zeta_k, \Theta_k)}{p(\zeta_i, L_i, \Theta_i)p(\zeta_j, L_j, \Theta_j)}, \quad (8.41)$$

Where $q(R_i, R_j)$ is an *affinity measure* [175] of the similarity of the two regions R_i and R_j (it is a weighted sum of the intensity difference $|\bar{I}_i - \bar{I}_j|$ and the chi-squared difference between the intensity histograms), $q(L_i)$ is given by the priors on the shape descriptors, and $q(\zeta_i, \Theta_i)$ is obtained by clustering in parameter space (see [193]).

Jump II: Switching Node Attributes

These moves switch the attributes of a node i . This involves changing the region type ζ_i and the model parameters Θ_i .

The move transitions between two states:

$$W = ((\zeta_i, L_i, \Theta_i), W_-) \rightleftharpoons W' = ((\zeta'_i, L'_i, \Theta'_i), W_-)$$

The proposal, see equation (8.37), should approximate:

$$\frac{p(W'|\mathbf{I})}{p(W|\mathbf{I})} = \frac{p(\mathbf{I}_{R_i}|\zeta'_i, L'_i, \Theta'_i)p(\zeta'_i, L'_i, \Theta'_i)}{p(\mathbf{I}_{R_i}|\zeta_i, L_i, \Theta_i)p(\zeta_i, L_i, \Theta_i)}.$$

We approximate this by a weight $\omega_4^{(\mu)}$ given by

$$\omega_4^{(\mu)} = \frac{q(L'_i)q(\zeta'_i, \Theta'_i)}{p(\mathbf{I}_{R_i}|\zeta_i, L_i, \Theta_i)p(\zeta_i, L_i, \Theta_i)},$$

where $q(L'_i)q(\zeta'_i, \Theta'_i)$ are the same functions used in the split and merge moves. The proposal probability is the weight normalized in the candidate set, $\mathbf{Q}_4(\mathbf{W}'|\mathbf{W} : \mathbf{I}) = \frac{\omega_4(\mathbf{W}')}{\sum_{\mu=1}^{\mathbf{N}_4} \omega_4^{(\mu)}}$.

AdaBoost for discriminative probabilities for face and text

This section describes how we use AdaBoost techniques to compute discriminative probabilities for detecting faces and text (strings of letters). We also describe the binarization algorithm used to detect the boundaries of text characters.

Computing discriminative probabilities by Adaboost

The standard AdaBoost algorithm, for example for distinguishing faces from non-faces [199], learns a binary-valued *strong classifier* H_{Ada} by combining a set of n binary-valued “weak classifiers” or feature tests $\text{Tst}_{\text{Ada}}(\mathbf{I}) = (h_1(\mathbf{I}), \dots, h_n(\mathbf{I}))$ using a set of weights $\boldsymbol{\alpha}_{\text{Ada}} = (\alpha_1, \dots, \alpha_n)$ [67],

$$H_{\text{Ada}}(\text{Tst}_{\text{Ada}}(\mathbf{I})) = \text{sign}\left(\sum_{i=1}^n \alpha_i h_i(\mathbf{I})\right) = \text{sign} \langle \boldsymbol{\alpha}_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}) \rangle. \quad (8.42)$$

The features are selected from a pre-designed dictionary Δ_{Ada} . The selection of features and the tuning of weights are posed as a supervised learning problem. Given a set of labeled examples, $\{(\mathbf{I}_i, \ell_i) : i = 1, 2, \dots, M\}$ ($\ell_i = \pm 1$), AdaBoost learning can be formulated as greedily optimizing the following function [173]

$$(\boldsymbol{\alpha}_{\text{Ada}}^*, \text{Tst}_{\text{Ada}}^*) = \arg \min_{\text{Tst}_{\text{Ada}} \subset \Delta_{\text{Ada}}} \arg \min_{\boldsymbol{\alpha}_{\text{Ada}}} \sum_{i=1}^M \exp^{-\ell_i \langle \boldsymbol{\alpha}_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}_i) \rangle}. \quad (8.43)$$

To obtain discriminative probabilities we use a theorem [68] which states that the features and test learnt by AdaBoost give (asymptotically) posterior probabilities for the object labels (e.g. face or non-face). The AdaBoost strong classifier can be rederived as the log posterior ratio test.

Theorem 8.3. (Friedman et al 1998) *With sufficient training samples M and features n , AdaBoost learning selects the weights $\boldsymbol{\alpha}_{\text{Ada}}^*$ and tests $\text{Tst}_{\text{Ada}}^*$ to satisfy*

$$q(\ell = +1|\mathbf{I}) = \frac{e^{\ell \langle \boldsymbol{\alpha}_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}) \rangle}}{e^{\langle \boldsymbol{\alpha}_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}) \rangle} + e^{-\langle \boldsymbol{\alpha}_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}) \rangle}}.$$

Moreover, the strong classifier converges asymptotically to the posterior probability ratio test

$$H_{\text{Ada}}(\text{Tst}_{\text{Ada}}(\mathbf{I})) = \text{sign}(\langle \boldsymbol{\alpha}_{\text{Ada}}, \text{Tst}_{\text{Ada}}(\mathbf{I}) \rangle) = \text{sign}\left(\frac{q(\ell = +1|\mathbf{I})}{q(\ell = -1|\mathbf{I})}\right).$$

In practice, the AdaBoost classifier is applied to windows in the image at different scales. Each window is evaluated as being face or non-face (or text versus non-text). For most images the posterior probabilities for faces or text are negligible for almost all parts of an image. So we use a cascade of tests [199,206] which enables us to rapidly reject many windows by setting their marginal probabilities to be zero.

Of course, AdaBoost will only converge to approximations to the true posterior probabilities $p(\ell|\mathbf{I})$ because only a limited number of tests can be used (and there is only a limited amount of training data).

Note that AdaBoost is only one way to learn a posterior probability, see theorem (8.1). It has been found to be very effective for object patterns which have relatively rigid structures, such as faces and text (the shapes of letters are variable but the patterns of a sequence are fairly structured [32]).

AdaBoost Training

We used standard AdaBoost training methods [67,68] combined with the cascade approach using asymmetric weighting [199,206]. The cascade enables the algorithm to rule out most of the image as face, or text, locations with a few tests and allows computational resources to be concentrated on the more challenging parts of the images.



Figure 8.28: Some scenes from which the training text patches are extracted.

The AdaBoost for text was designed to detect text segments. Our test data was extracted by hand from 162 images of San Francisco, see Figure 8.28, and contained 561 text images. More than half of the images were taken by blind volunteers (which reduces bias). We divided each text image into several overlapping text segments with fixed width-to-height ration 2:1 (typically containing between two and three letters). A total of 7,000 text segments were used as the positive training set. The negative examples were obtained by a bootstrap process similar to Drucker et al [58]. First we selected negative examples by randomly sampling from windows in the image dataset. After training with these samples, we applied the AdaBoost algorithm at a range of scales to classify all windows in the training images. Those misclassified as text were then used as negative examples for the next stage of AdaBoost. The image regions most easily confused with text were vegetation, and repetitive structures such as railings or building facades. The features used for AdaBoost were image tests corresponding to the statistics of elementary filters. The features were chosen to detect properties of text segments that were relatively invariant to the shapes of the individual letters or digits. They included averaging the intensity within image windows, and statistics of the number of edges. We refer to [32] for more details.

The AdaBoost posteriors for faces was trained in a similar way. This time we used Haar basis functions [199] as elementary features. We used the FERET [159] database for our positive examples,

and by allowing small rotation and translation transformation we had 5,000 positive examples. We used the same strategy as described above for text to obtain negative examples.

In both cases, we evaluated the log posterior ratio test on testing datasets using a number of different thresholds (see [199]). In agreement with previous work on faces [199], AdaBoost gave very high performance with very few false positives and false negatives, see table (8.1). But these low error rates are slightly misleading because of the enormous number of windows in each image, see table (8.1). A small false positive rate may imply a large number of false positives for any regular image. By varying the threshold, we can either eliminate the false positives or the false negatives but not both at the same time. We illustrate this by showing the face regions and text regions proposed by AdaBoost in Figure 8.29. If we attempt classification by putting a threshold then we can only correctly detect all the faces and the text at the expense of false positives.

Object	False Positive	False Negative	Images	Subwindows
Face	65	26	162	355,960,040
Face	918	14	162	355,960,040
Face	7542	1	162	355,960,040
Text	118	27	35	20,183,316
Text	1879	5	35	20,183,316

Table 8.1: Performance of AdaBoost at different thresholds.



Figure 8.29: The boxes show faces and text as detected by the AdaBoost log posterior ratio test with fixed threshold. Observe the false positives due to vegetation, tree structure, and random image patterns. It is impossible to select a threshold which has no false positives and false negatives for this image. As it is shown in our experiments later, the generative models will remove the false positives and also recover the missing text.

When Adaboost is integrated with the generic region models in the image parser, the generic region proposals can remove false positives and find text that AdaBoost misses. For example, the '9' in the right panel of Figure 8.29 is not detected because our AdaBoost algorithm was trained on text segments. Instead it is detected as a generic shading region and later recognized as a letter '9', see Figure 8.31. Some false positive text and faces in figure 8.29 are removed in Figures 8.31 and 8.33.

The AdaBoost algorithm for text needs to be supplemented with a binarization algorithm, described below, to determine text character location. This is followed by applying shape contexts

[13] and informative features [192] to the binarization results to make proposals for the presence of specific letters and digits.

In many cases, see Figure 8.30, the results of binarization are so good that the letters and digits can be detected immediately (i.e. the proposals made by the binarization stage are automatically accepted). But this will not always be the case. We note that binarization gives far better results than alternatives such as edge detection [29].



Figure 8.30: Example of binarization on the detected text.

The binarization algorithm is a variant of one proposed by Niblack [151]. We binarize the image intensity using an adaptive thresholding based on a adaptive window size. Adaptive methods are needed because image windows containing text often have shading, shadow, and occlusion. Our binarization method determines the threshold $T_b(v)$ for each pixel v by the intensity distribution of its local window $r(v)$ (centered on v).

$$T_b(v) = \mu(\mathbf{I}_{r(v)}) + k \cdot std(\mathbf{I}_{r(v)}),$$

where $\mu(\mathbf{I}_{r(v)})$ and $std(\mathbf{I}_{r(v)})$ are the intensity mean and standard deviation within the local window. The size of the local window is selected to be the smallest possible window whose intensity variance is above a fixed threshold. The parameter $k = \pm 0.2$, where the \pm allows for cases where the foreground is brighter or darker than the background.

8.8.6 Image Parsing Experiments

The image parsing algorithm is applied to a number of outdoor/indoor images. The speed in PCs (Pentium IV) is comparable to segmentation methods such as normalized cuts [132] or the DDMCMC algorithm in [193]. It typically runs around 10-40 minutes. The main portion of the computing time is spent at segmenting generic regions and boundary diffusion [224].

Figures 8.31, 8.32, and 8.33 show some challenging examples which have heavy clutter and shading effects. We present the results in two parts. One shows the segmentation boundaries for generic regions and objects, and the other shows the text and faces detected with text symbols to indicate text recognition, i.e. the letters are correctly read by the algorithm. Then we synthesize images sampled from the likelihood model $p(\mathbf{I}|W^*)$ where W^* is the parsing graph (the faces, text, regions parameters and boundaries) obtained by the parsing algorithm. The synthesized images are used to visualize the parsing graph W^* , i.e. the image content that the computer “understand”.

In the experiments, we observed that the face and text models improved the image segmentation results by comparison to our previous work [193] which only used generic region models. Conversely, the generic region models improve object detection by removing some false alarms and recovering objects which were not initially detected. We now discuss specific examples.

In Figure 8.29, we showed two images where the text and faces were detected purely bottom-up using AdaBoost. It was impossible to select a threshold so that our AdaBoost algorithm had no false positives or false negatives. To ensure no false negatives, apart from the ‘9’, we had to lower

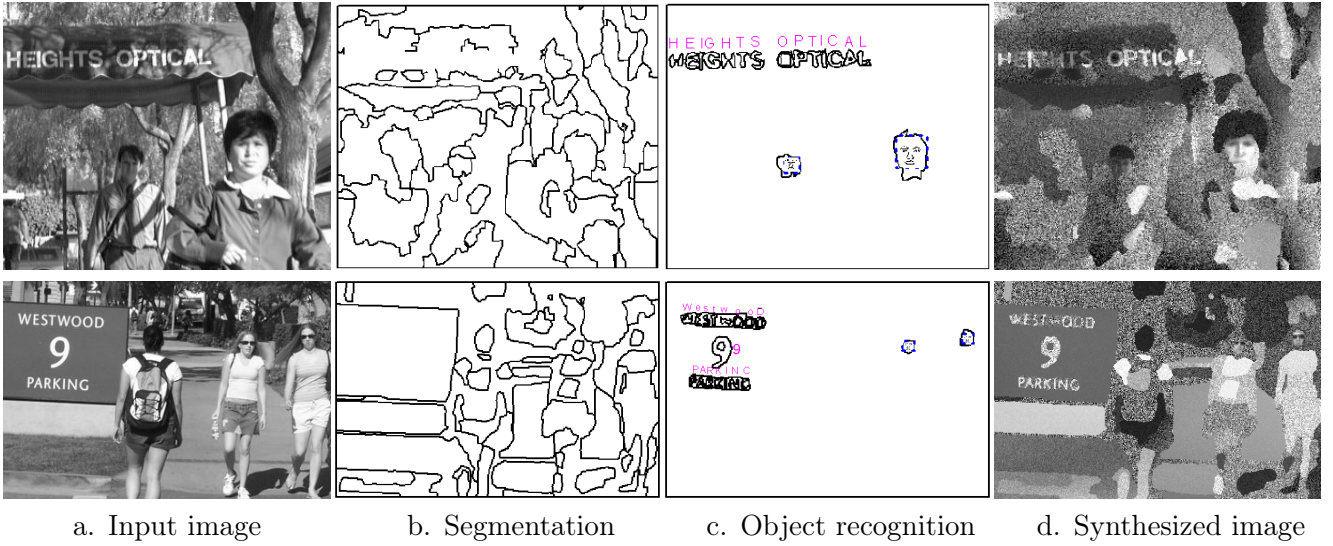


Figure 8.31: Results of segmentation and recognition on two images. The results are improved compare to the purely bottom-up (AdaBoost) results displayed in Figure 8.29.

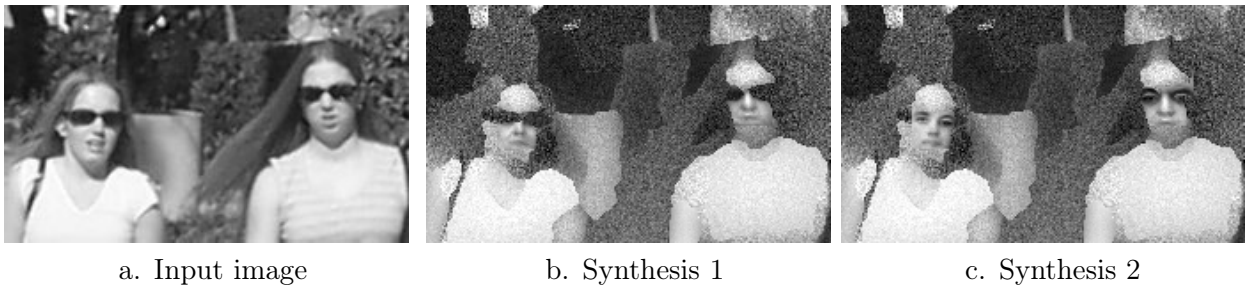


Figure 8.32: A close-up look of an image in Figure 8.31. The dark glasses are explained by the generic shading model and so the face model does not have to fit this part of the data. Otherwise the face model would have difficulty because it would try to fit the glasses to eyes. Standard AdaBoost only correctly classifies these faces at the expense of false positives, see Figure 8.29. We show two examples of synthesized faces, one (Synthesis 1) with the dark glasses (modelled by shading regions) and the other (Synthesis 2) with the dark glasses removed (i.e. using the generative face model to sample parts of the face (e.g. eyes) obscured by the dark glasses).

the threshold and admit false positives due to vegetation and heavy shadows (e.g. the shadow in the sign “HEIGHTS OPTICAL”).

The letter ‘9’ was not detected at any threshold. This is because our AdaBoost algorithm was trained to detect text segments, and so did not respond to a single digit.

By comparison, Figure 8.31 shows the image parsing results for these two images. We see that the false alarms proposed by AdaBoost are removed because they are better explained by the generic region models. The generic shading models help object detection by explaining away the heavy shading on the text “HEIGHTS OPTICAL” and the dark glasses on the women, see Figure 8.32. Moreover, the missing digit ‘9’ is now correctly detected. The algorithm first detected it as a generic shading region and then reclassified as a digit using the sub-kernel that switches node attributes.

The ability to synthesize the image from the parsing graph W^* is an advantage of the Bayesian

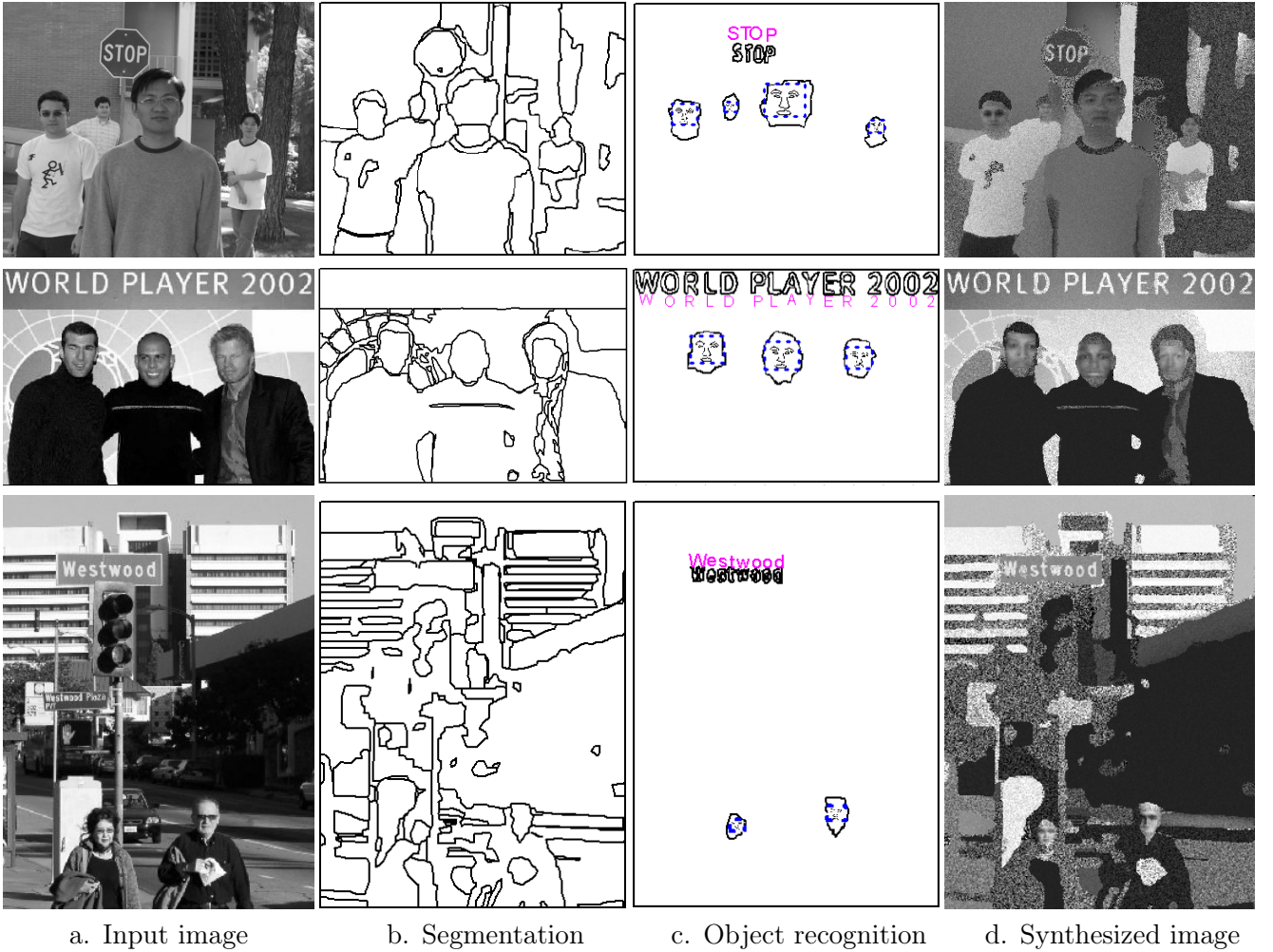


Figure 8.33: Results of segmentation and recognition on outdoor images. Observe the ability to detect faces and text at multiple scale.

approach. The synthesis helps illustrate the successes, and sometimes the weaknesses, of the generative models. Moreover, the synthesized images show how much information about the image has been captured by the models. In table (8.2), we give the number of variables used in our representation W^* and show that they are roughly proportional to the jpeg bytes. Most of the variables in W^* are used to represent points on the segmentation boundary, and at present they are counted independently. We could reduce the coding length of W^* substantially by encoding the boundary points effectively, for example, using spatial proximity. Image encoding is not the goal of our current work, however, and more sophisticated generative models would be needed to synthesize very realistic images.

In this section, we describe two challenging technical problems for image parsing. Our current work addresses these issues.

1. Two mechanisms for constructing the parsing graph

In the introduction to this section we stated that the parsing graph can be constructed in compositional and decompositional modes. The compositional mode proceeds by grouping small elements while the decompositional approach involves detecting an object as a whole and then locating its parts, see Figure 8.34.

Image	Stop	Soccer	Parking	Street	Westwood
jpg bytes	23,998	19,563	23,311	26,170	27,790
$ W^* $	4,886	3,971	5,013	6,346	9,687

Table 8.2: The number of variables in W^* for each image compared to the JPG bytes.

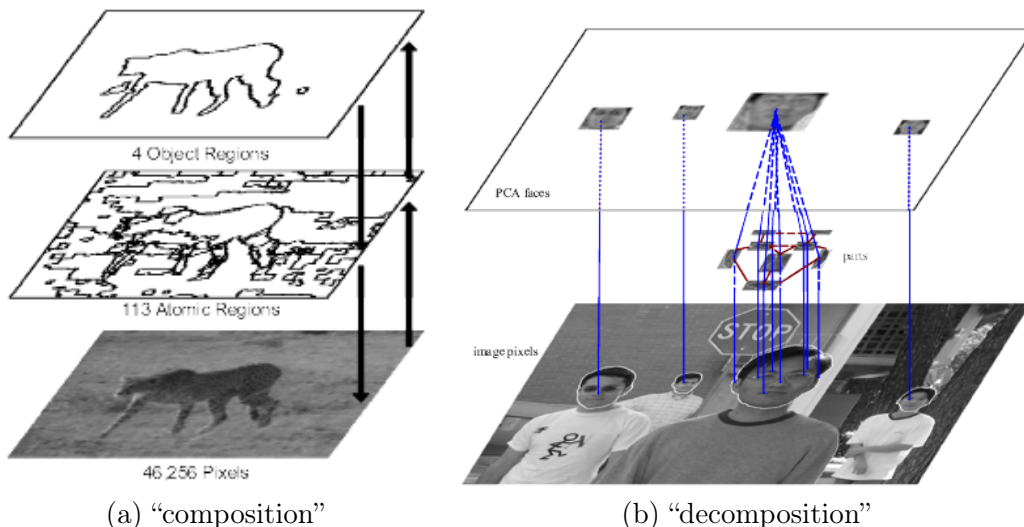


Figure 8.34: Two mechanisms for constructing the parsing graph. See text for explanation.

The compositional mode appears most effective for Figure 8.34(a). Detecting the cheetah by bottom-up tests, such as those learnt by AdaBoost, seems difficult owing to the large variability of shape and photometric properties of cheetahs. By contrast, it is quite practical using Swendsen-Wang Cuts [8] to segment the image and obtain the boundary of the cheetah using a bottom-up compositional approach and a parsing tree with multiple levels. The parsing graph is constructed starting with the pixels as leaves (there are 46,256 pixels in Figure 8.34(a)). The next level of the graph is obtained using local image texture similarities to construct graph nodes (113 of them) corresponding to “atomic regions” of the image. Then the algorithm constructs nodes (4 of them) for “texture regions” at the next level by grouping the atomic regions (i.e. each atomic region node will be the child of a texture region node). At each level, we compute a discriminative (proposal) probability for how likely adjacent nodes (e.g. pixels or atomic regions) belong to the same object or pattern. We then apply a transition kernel implementing split and merge dynamics (using the proposals). We refer to [8] for more detailed discussion.

For objects with little variability, such as the faces shown in Figure 8.34(b), we can use bottom-up proposals (e.g. AdaBoost) to activate a node that represents the entire face. The parsing graph can then be constructed downwards (i.e. in the decompositional mode) by expanding the face node to create child nodes for the parts of the face. These child nodes could, in turn, be expanded to grandchild nodes representing finer scale parts. The amount of node expansion can be made adaptive to depend on the resolution of the image. For example, the largest face in Figure 8.34(b) is expanded into child nodes but there is not sufficient resolution to expand the face nodes corresponding to the three smaller faces.

The major technical problem is to develop a mathematical criterion for which mode is most effective for which types of objects and patterns. This will enable the algorithm to adapt its search strategy accordingly.

2. Optimal ordering strategy for tests and kernels

The control strategy of our current image parsing algorithm does not select the tests and sub-kernels in an optimal way. At each time step, the choice of sub-kernel is independent of the current

state W (though the choice of where in the graph to apply the sub-kernel will depend on W). Moreover, bottom-up tests are performed which are never used by the algorithm.

It would be more efficient to have a control strategy which selects the sub-kernels and tests adaptively, provided the selection process requires low computational cost. We seek to find an optimal control strategy for selection which is effective for a large set of images and visual patterns. The selection criteria should select those tests and sub-kernels which maximize the gain in information.

We propose the two information criteria. The first is stated in Theorem 8.1. It measures the information gained for variable w in the parsing graph by performing a new test Tst_+ . The information gain is $\delta(w|\text{Tst}_+) = KL(p(w|\mathbf{I}) || q(w|\text{Tst}(\mathbf{I}))) - KL(p(w|\mathbf{I}) || q(w|\text{Tst}_t(\mathbf{I}), F_+))$, where $\text{Tst}(\mathbf{I})$ denotes the previous tests (and KL is the Kullback-Leibler divergence).

The second is stated in Theorem 8.2. It measures the power of a sub-kernel \mathcal{K}_a by the decrease of the KL-divergence $\delta(\mathcal{K}_a) = KL(p || \mu_t) - KL(p || \mu_t \mathcal{K}_a)$. The amount of decrease δ_a gives a measure of the power of the sub-kernel \mathcal{K}_a when informed by $\text{Tst}_t(\mathbf{I})$.

We need also take into account the computational cost of the selection procedures. See [20] for a case study for how to optimally select tests taking into account their computational costs.

Chapter 9

Hamiltonian Monte Carlo

Hamilton Monte Carlo (originally known as Hybrid Monte Carlo, and referred to throughout this chapter as HMC) is a powerful framework for sampling in continuous spaces. Like many other MCMC methods (slice sampling, SW sampling, ...), HMC is an auxiliary variable method, which means that a set of helper variables is introduced to facilitate movement in the original space. In HMC, the original set of variables are treated as "position" variables in the energy landscape, and the auxiliary variables give the "momentum" in the position dimensions. Each position dimension has a single corresponding momentum variable, so the joint space of the original and auxiliary variable has twice as many dimensions as the original space. Once the momentum variables have been introduced, Hamiltonian mechanics is used to simulate the time evolution of the physical system defined by the position and momenta, allowing for movement in the joint space of the original and auxiliary variables in a way that preserves the distribution in the original space.

9.1 Introduction to Hamiltonian Mechanics

To understand the HMC sampling method, it is necessary to have some knowledge of the basic principles of Hamiltonian Mechanics. Hamiltonian Mechanics is an alternative formulation of Lagrangian mechanics, and it is equivalent to both Lagrangian Mechanics and Newtonian Mechanics. In Hamiltonian Mechanics, any configuration of a physical system can be expressed by a pair of n -dimensional variables q and p , which respectively represent the position and momentum of the physical system at a single point in time. In simple cases, position and momentum have their usual interpretation, but in general they should be thought of as abstract quantities representing the degrees of freedom in a system. The behavior of the physical system is governed by a function $H(q, p)$, called the Hamiltonian, which represents the energy of the system. In some cases, the Hamiltonian can be written in the form $H(q, p) = U(q) + K(p)$, where $U(q)$ represents the potential energy of the system and $K(p)$ represents the kinetic energy.

As a simple example of Hamiltonian Mechanics, imagine a point mass moving in a hilly and frictionless landscape. Any state of this system can be described by a pair (q, p) , where q is a 2-dimensional variable giving the position of the point mass (latitude and longitude coordinates) and p is a 2-dimensional variable giving the momentum in each direction. The Hamiltonian can be written as $H(q, p) = U(q) + K(p)$, where $U(q)$ is proportional to the height of the point mass relative to some reference point and $K(p) = ||p||^2/(2m)$, where m is the mass.

The movement of the point mass will be determined by the tradeoff between kinetic and potential energy that occurs as it travels through the landscape. For example, a state with no momentum on the side of a slope will be pulled downward, and its potential energy will be transferred into kinetic energy moving towards the bottom of the slope. On the other hand, if the point mass is moving forward along a flat plain and then encounters a barrier, kinetic energy is transferred to potential energy and the mass will slow or even reverse direction back towards the plain if the barrier is sufficiently steep.

This simple model is quite similar to the "physical system" of Hamiltonian Monte Carlo. In HMC, the position q represents a point in the parameter space of the target density (n -dimensional instead of 2-dimensional), and $U(q)$ is the "potential energy" of the point q given by the negative log density at q . The momentum p in HMC is a normally distributed variable with an energy function of the form $p^\top \Sigma^{-1} p$ for some positive definite covariance matrix Σ . The simplest choice is $\Sigma = \sigma^2 I_n$, which gives the standard kinetic energy equation when $\sigma = \sqrt{m}$. The joint state (q, p) is then evolved in time to reach a proposal state (q^*, p^*) in a way intuitively similar to a point mass moving through a frictionless hilly landscape. The introduction of the auxiliary momentum p ultimately allows for movement in the target density space of q .

The time evolution of the physical system is given by a pair of partial differential equations known as Hamilton's Equations, which are

$$\frac{dq}{dt} = \frac{\partial H}{\partial p}, \quad (9.1)$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q}. \quad (9.2)$$

Hamilton's equations can be derived by taking a Legendre transform of the Lagrangian of the physical system. Updating q and p according to Hamilton's Equations ensures the conservation of many properties of properties of the system, including energy, so that the value of the Hamiltonian $H(q, p)$ should remain constant over time.

Consider the two scenarios of the point mass mentioned earlier, in the first case moving down a slope and in the second case encountering a barrier while moving along a flat plain. These two scenarios have a natural interpretation in HMC. The first corresponds to the optimization phase of sampling, which occurs when the current state is far away from a local minima. In this case, HMC behaves similarly to gradient descent, since movement towards a local minima is encouraged. The second corresponds to the case when the current state is already near a local minima, and the auxiliary momentum variables allow movement in the local minima basin. In some cases, the momentum may be able to overcome local obstacles and find other local minima in the energy landscape, and in some cases the chain may simply become stuck in a strong energy basin, restricted to sampling from a single mode. If the mode local mode is tightly constrained in some linear directions (similar to a canyon in the point mass example), then movement through the mode will require momenta in the unconstrained directions.

9.2 Properties of Hamiltonian Mechanics

Hamiltonian Mechanics has several important properties which ensure that HMC satisfies detailed balance and preserves the target distribution. These properties will be discussed from a continuous viewpoint first, and from the viewpoint of discrete numeric implementation in the next section.

9.2.1 Conservation of Energy

Updating a physical system with Hamilton's equations preserves the value of the Hamiltonian $H(q, p)$, so that the value of $H(q, p)$ should remain constant over time, even though q and p will vary. The proof of this property is straightforward:

$$\frac{dH}{dt} = \sum_{i=1}^n \left[\frac{\partial H}{\partial q_i} \frac{dq_i}{dt} + \frac{\partial H}{\partial p_i} \frac{dp_i}{dt} \right] = \sum_{i=1}^n \left[\frac{\partial H}{\partial q_i} \frac{\partial H}{\partial p_i} - \frac{\partial H}{\partial q_i} \frac{\partial H}{\partial p_i} \right] = 0.$$

This property is important in HMC because it ensures that $H(q, p) = H(q^*, p^*)$, where (q, p) is the previous state in the joint space and (q^*, p^*) is the proposed state. Combined with the other properties of Hamiltonian Mechanics, conservation of energy can be used to prove that ideal HMC defines a reversible Metropolis-Hastings proposal with an acceptance probability of 1, as will be

shown in a later section. In practice, this property is only approximately true, since discrete numeric approximations must be used to solve Hamilton's Equations, so that $H(q, p)$ might differ from $H(q^*, p^*)$. If the numeric approximation is accurate, this difference should be relatively small, and high acceptance probabilities can still be achieved.

9.2.2 Reversibility

The mapping from $(q(t), p(t))$ to $(q(t + s), p(t + s))$ defined by Hamiltonian Mechanics is unique and therefore can be inverted. If $H(q, p) = U(q) + K(p)$ and $K(p) = K(-p)$, which is true in HMC when Gaussian auxiliary variables centered at 0 are used, then the inverse mapping can be given explicitly by negating p at the end of the path, evolving the system for the same time s , and then negating p once more. Reversibility will be used to show that HMC satisfies detailed balance, which is the simplest way to prove that an MCMC method preserves the correct distribution. Reversibility can be exactly preserved in discrete implementation of Hamilton's Equations.

9.2.3 Symplectic Structure and Volume Preservation

For any smooth function $H : \mathbb{R}^{2n} \rightarrow \mathbb{R}$, Hamilton's Equations define a special type of vector field and a symplectic structure on the manifold \mathbb{R}^{2n} . A symplectic manifold is a smooth manifold M (in practice, usually \mathbb{R}^{2n}) with a differential 2-form ω called the symplectic form. The standard symplectic form used in \mathbb{R}^{2n} is $\omega = \begin{pmatrix} 0 & I_n \\ -I_n & 0 \end{pmatrix}$, which is related to Hamilton's Equations, since

$$\frac{d}{dt}(q, p) = \omega \frac{dH}{d(q, p)}$$

In general, ω only needs to be closed and non-degenerate 2-form on M . The symplectic form can be intuitively understood as a way of generating a vector field from the differential 1-form dH of a Hamiltonian energy function $H(q, p)$.

The solution obtained from integrating Hamilton's Equations (or equivalently the flow induced by the vector field of the Hamiltonian H on the symplectic manifold M) has the important property of preserving the symplectic form ω . In other words, the mapping $(q(t), p(t)) \mapsto (q(t + s), p(t + s))$ over all $(q, p) \in M$ defines a diffeomorphism from M to itself that respects the structure of ω . The invariance of ω under Hamiltonian flows is the mathematical foundation of the many conservation properties of Hamiltonian Mechanics, including conservation of energy.

An important consequence of the conservation of the 2-form ω is the conservation of volume under Hamiltonian flows, a result known as *Louisville's Theorem*. Using symplectic geometry, the proof of this theorem is quite simple. The non-degenerate 2-form ω on M can be raised to the n^{th} power to define a non-degenerate volume form ω^n (ω^n is a $2n$ -form, since ω is a 2-form), and the conservation of ω under Hamiltonian flows implies the conservation of the volume form ω^n . This property is important for HMC because it ensures that the change of coordinates $(q, p) \mapsto (q^*, p^*)$ obtained by updating the joint state according to Hamilton's equations has a Jacobian with a determinant that equals 1 in absolute value. Without volume preservation, the difficulty of calculating the determinant of the Jacobian of the coordinate change to rescale the density when proposing a move to (q^*, p^*) would likely be a large enough impediment to make HMC infeasible in practice. Volume preservation can hold exactly even in discrete implementations of Hamilton's Equations if the correct updating scheme is used.

A simple proof of volume preservation can be given using only Hamilton's Equations without reference to symplectic geometry. Let $V = \left(\frac{dq}{dt}, \frac{dp}{dt} \right)$ be the vector field defined by Hamilton's Equations. Then the divergence of V is 0 everywhere, because

$$\text{div}(V) = \sum_{i=1}^n \left(\frac{\partial}{\partial q_i} \frac{dq_i}{dt} + \frac{\partial}{\partial p_i} \frac{dp_i}{dt} \right) = \sum_{i=1}^n \left(\frac{\partial}{\partial q_i} \frac{\partial H}{\partial p_i} - \frac{\partial}{\partial p_i} \frac{\partial H}{\partial q_i} \right) = \sum_{i=1}^n \left(\frac{\partial^2 H}{\partial q_i \partial p_i} - \frac{\partial^2 H}{\partial q_i \partial p_i} \right) = 0,$$

and a vector field with divergence 0 can be shown to preserve volume.

9.3 The Leapfrog Discretization of Hamilton's Equations

It is impossible to solve Hamilton's Equations exactly except in the simplest systems, so numeric implementations of Hamiltonian Mechanics must rely on a discrete approximation to the true continuous solution defined by the equations. Before discussing the most effective and widely used discretization, known as the Leapfrog Integrator, two less effective but instructive methods are introduced.

9.3.1 Euler's Method and Modified Euler's Method

The most naive method of discretizing the time evolution of a Hamiltonian H under Hamilton's Equations is to update q and p simultaneously by some small step size ε as follows:

$$\begin{aligned} p(t + \varepsilon) &= p(t) + \varepsilon \frac{dp}{dt}(q(t), p(t)) = p(t) - \varepsilon \frac{\partial H}{\partial q}(q(t), p(t)), \\ q(t + \varepsilon) &= q(t) + \varepsilon \frac{dq}{dt}(q(t), p(t)) = q(t) + \varepsilon \frac{\partial H}{\partial p}(q(t), p(t)). \end{aligned}$$

This discretization is known as Euler's Method. It does not preserve volume and can lead to inaccurate approximations after only a few steps.

An improvement of Euler's Method is the Modified Euler's Method, which uses alternating ε -size updates of the q and p . If the energy function can be written in the form $H(q, p) = U(q) + K(p)$, then updating q only depends on p and vice versa, because

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = \frac{\partial K}{\partial p} \quad \text{and} \quad \frac{dp}{dt} = -\frac{\partial H}{\partial q} = -\frac{\partial U}{\partial q}. \quad (9.3)$$

One step the Modified Euler's Method consists of updating the current q from the current p with step size ε , and then updating the current p from the *updated* q with the same step size ε (the reverse order of updating p then q is equally valid) as follows:

$$\begin{aligned} p(t + \varepsilon) &= p(t) - \varepsilon \frac{\partial U}{\partial q}(q(t)), \\ q(t + \varepsilon) &= q(t) + \varepsilon \frac{\partial K}{\partial p}(p(t + \varepsilon)). \end{aligned}$$

These alternating updates are shear transformation, and therefore preserve volume just like the true continuous solution to Hamilton's Equations. Shear transformations have a Jacobian with a determinant which has absolute value 1, so these updates can be performed with no extra computational cost in the HMC algorithm. The Modified Euler discretization is much more accurate and stable than the original Euler's Method.

However, the Modified Euler's Method is not reversible because of the order of the updates. Suppose that you have chosen to update q and then p , although the opposite order is equally valid. After applying some number of updates and reaching a new state, one could invert the mapping by negating p , letting the dynamics unfold, and negating p again at the end of the trajectory, but to reach exactly the same state, it would be necessary to use the *opposite* update order p and then q at each step to undo the forward updates. Trying to reverse the path by updating in the original order q then p would likely result in a state close to the original state, but discretization error prevents this reversal from being exact. An ideal integration scheme should be able to exactly reverse all updates.

9.3.2 The Leapfrog Integrator

The Leapfrog Integrator is a close relative of the Modified Euler's Method, and it is the standard discrete integration scheme used in HMC because it is straightforward to implement and it exactly satisfies both volume preservation and reversibility, which are desirable properties of the true continuous solution to Hamilton's Equations. The Leapfrog Integrator only satisfies these properties when the Hamiltonian has the form $H(q, p) = U(q) + K(p)$, in which case the equations in (9.3) hold. In standard HMC practice, the variables p are normally distributed, and p has the quadratic energy function $K(p) = p^\top \Sigma^{-1} p / 2$ with the derivative $\frac{\partial K}{\partial p}(p) = \Sigma^{-1} p$ for a p.d. covariance matrix Σ . A single step of size of the Leapfrog Integrator is given below, where ε is a parameter for step size:

$$p(t + \varepsilon/2) = p(t) - (\varepsilon/2) \frac{\partial U}{\partial q}(q(t)), \quad (9.4)$$

$$q(t + \varepsilon) = q(t) + \varepsilon \frac{\partial K}{\partial p}(p(t + \varepsilon/2)), \quad (9.5)$$

$$p(t + \varepsilon) = p(t + \varepsilon/2) - (\varepsilon/2) \frac{\partial U}{\partial q}(q(t + \varepsilon)). \quad (9.6)$$

A Leapfrog update consists of a $(\varepsilon/2)$ -size update of p with the old q , followed by a ε -size update of q with the new p , followed by a $(\varepsilon/2)$ -size update of p with the new q . When performing multiple Leapfrog steps, the above scheme is equivalent to performing half-step updates of p only at the very beginning and very end of the trajectory, and alternating between full step updates of q and p in between, since the two $(\varepsilon/2)$ -size updates of p at the end of an old step and beginning of a new step are equivalent to a single ε -size update of p .

The only difference between the Modified Euler's Method and the Leapfrog Method is the splitting of the tail-end full step p -update in a Modified Euler trajectory into two half-step p -updates at the beginning and the end of a Leapfrog trajectory.

9.3.3 Properties of the Leapfrog Integrator

The symmetry of the Leapfrog Integrator ensures reversibility, because a single Leapfrog step can be reversed by negating p , applying the Leapfrog Integrator, and negating p again. The negation of p at the end of a trajectory is needed for reversibility in HMC as well, but can be ignored in standard HMC practice where Gaussian auxiliary variables are used because $K(p) = K(-p)$ when p is Gaussian.

The Leapfrog Integrator is volume-preserving for the same reason as the Modified Euler's Method: since the updates of q only depend on p and vice-versa, the change of coordinates defined by a Leapfrog step is a composition of three volume-preserving shear transformations with a Jacobian of determinant 1, which define a single coordinate change with Jacobian determinant 1, since the Jacobian determinant of the composition of coordinate changes is the product of the Jacobian determinants of the individual coordinate changes.

An informal proof that the mapping $(q(t), p(t)) \mapsto (q(t), p(t + \varepsilon/2))$ defined by Equation (9.4) is a shear transformation is given below, and an almost identical proof can be used to show that Equations (9.5) and (9.6) are also shear transformations. The key property is that updating p depends only on a smooth function of q , or vice-versa, for each step of a Leapfrog update, which

holds because $H(q, p) = U(q) + K(p)$. Let $J_p = \begin{pmatrix} \frac{\partial q^*}{\partial q} & \frac{\partial q^*}{\partial p} \\ \frac{\partial p^*}{\partial q} & \frac{\partial p^*}{\partial p} \end{pmatrix}$ be the Jacobian of the coordinate change $(q, p) \mapsto (q^*, p^*)$ corresponding to $(q(t), p(t)) \mapsto (q(t), p(t + \varepsilon/2))$. Consider some initial state $(q(0), p(0))$ and its close neighbor $(q'(0), p'(0)) = (q(0) + \delta u_q, p(0) + \delta u_p)$ for some unit vector $u = (u_q, u_p)$ and some small $\delta > 0$. The first step of the Leapfrog update for these two states is given by

$$p(\varepsilon/2) = p(0) - (\varepsilon/2) \frac{\partial U}{\partial q}(q(0)),$$

$$p'(\varepsilon/2) = p'(0) - (\varepsilon/2) \frac{\partial U}{\partial q}(q'(0)) = p(0) + \delta u_p - (\varepsilon/2) \frac{\partial U}{\partial q}(q(0) + \delta u_q),$$

and $q(\varepsilon/2) = q(0)$, $q'(\varepsilon/2) = q'(0) = q(0) + \delta u_q$ since q is not updated during this step. Using a Taylor Expansion, $\frac{\partial U}{\partial q}(q(0) + \delta u_q) \approx \frac{\partial U}{\partial q}(q(0)) + \delta [\frac{\partial^2 U}{\partial q^2}(q(0))]u_q$ for small δ . Therefore

$$\begin{pmatrix} q'(\varepsilon/2) - q(\varepsilon/2) \\ p'(\varepsilon/2) - p(\varepsilon/2) \end{pmatrix} \approx \delta \begin{pmatrix} I_n & 0 \\ -(\varepsilon/2) \frac{\partial^2 U}{\partial q^2}(q(0)) & I_n \end{pmatrix} \begin{pmatrix} u_q \\ u_p \end{pmatrix}$$

and letting δ go to 0 implies

$$J_p = \begin{pmatrix} \frac{\partial q^*}{\partial q} & \frac{\partial q^*}{\partial p} \\ \frac{\partial p^*}{\partial q} & \frac{\partial p^*}{\partial p} \end{pmatrix} = \begin{pmatrix} I_n & 0 \\ -(\varepsilon/2) \frac{\partial^2 U}{\partial q^2} & I_n \end{pmatrix}$$

which is a shear matrix with determinant 1. Note that ε is arbitrary and fixed in this proof, and that the limit is taken only in the spatial perturbation δ . The Leapfrog Integrator exactly preserves volume for any ε . The other two steps of a Leapfrog update can be shown to be shear transformations by using a Taylor Expansion of the derivative $\frac{\partial U}{\partial q}$ or $\frac{\partial K}{\partial p}$ in the same way. If a Gaussian auxiliary variable is used in HMC, the q -update given by Equation (9.5) has a Jacobian of the form

$$J_q = \begin{pmatrix} I_n & \varepsilon \Sigma^{-1} \\ 0 & I_n \end{pmatrix}$$

where Σ is the covariance matrix of p . Using a Gaussian proposal with $\Sigma \approx \frac{\partial^2 U}{\partial q^2}$ can dramatically improve movement through the q -space, especially when sampling from distributions with high ratios between the constrained width of the largest and smallest linear directions, i.e. a large ratio between the maximum and minimum eigenvalue of the local covariance. Unfortunately, if the energy function U does not have constant curvature, then Σ must vary with the position q , in which case the $H(q, p)$ is no longer separable, the Leapfrog Integrator does not preserve volume, and solving Hamilton's Equations becomes much more difficult. See the RMHMC section for a detailed discussion.

9.4 Hamiltonian Monte Carlo and Langevin Monte Carlo

HMC arises very naturally from Hamiltonian Mechanics because the properties of Hamiltonian Mechanics can be used in a straightforward way to show detailed balance for HMC, which ensures invariance of the stationary distribution after an HMC step. First the HMC method is introduced, along with a special case of HMC known as Langevin Monte Carlo (referred to throughout this section as LMC, also known as the Metropolis-Adjusted Langevin Algorithm or MALA), which has different properties and uses than "full" HMC. This is followed by a proof that HMC satisfies detailed balance and defines a valid sampling scheme and a discussion of tuning HMC.

9.4.1 Formulation of HMC

To be consistent with the notation used in the preceding sections, the target density to be sampled during HMC will be written as

$$P(q) = \frac{1}{Z} \exp \{-U(q)\} \quad (9.7)$$

for $q \in \mathbb{R}^n$ and a smooth potential energy function $U : \mathbb{R}^n \rightarrow \mathbb{R}$ with normalizing constant Z . For a discussion of handling constraints in HMC so that q can be restricted to a set $U \subset \mathbb{R}^n$,

see [148]. In Bayesian inference problems, U is the negative log of the posterior distribution for a set of parameters q and dataset X with a prior π and log-likelihood l , i.e.

$$U(q) = -\log[\pi(q)] - l(X|q).$$

HMC is an auxiliary variable method, and the standard auxiliary variables are $p \sim N(0, \Sigma)$ with negative log density

$$K(p) = \frac{1}{2} p^\top \Sigma^{-1} p \quad (9.8)$$

for some $n \times n$ positive-definite covariance matrix Σ . The pair $(q, p) \in \mathbb{R}^{2n}$ has the joint density

$$P(q, p) = \frac{1}{Z} \exp\{-H(q, p)\} = \frac{1}{Z} \exp\left\{-U(q) - \frac{1}{2} p^\top \Sigma^{-1} p\right\} \quad (9.9)$$

and the joint energy function

$$H(q, p) = U(q) + K(p) = U(q) + \frac{1}{2} p^\top \Sigma^{-1} p. \quad (9.10)$$

The joint density $P(q, p) = \frac{1}{Z} e^{-H(q, p)}$ has a marginal distribution $q \sim \frac{1}{Z} \exp^{-U(q)}$ because

$$\int_{\mathbb{R}^n} P(q, p) dp = \frac{1}{Z_q} e^{-U(q)} \int_{\mathbb{R}^n} \frac{1}{Z_p} e^{-\frac{1}{2} p^\top \Sigma^{-1} p} dp = \frac{1}{Z_q} e^{-U(q)}. \quad (9.11)$$

Therefore sampling from the joint energy $H(q, p)$ will provide a sample of q distributed according to the target energy $U(q)$.

Only normally distributed auxiliary variables will be discussed in this book, and normal auxiliary variables are the natural choice for several reasons. The standard kinetic energy function $K(p) = \|p\|^2/m$ used in physics is equivalent to (9.8) with $\Sigma = mI_n$. Normal distributions can be simulated accurately and efficiently, and can provide reasonable approximations of local manifold structure in high dimensional spaces (for example, PCA reconstructions of the human face). Normal distributions also have the property that $K(p) = K(-p)$, which makes the negation of p needed at the end of an HMC step ignorable.

In this section, Σ will be fixed, but successfully sampling from complex high-dimensional distributions likely requires a dependence $p \sim N(0, \Sigma(q))$ on the current state q in the target distribution so that useful updates of Hamilton's Equations can be achieved in each Leapfrog Step. In the simplest case, if $q \sim N(0, \Phi)$, then a proposal $p \sim N(0, \Phi^{-1})$ is ideal, because the q updates are scaled according to p and vice-versa. More generally, $\Sigma(q) = \frac{\partial^2 U}{\partial q^2}$ and $p \sim N(0, \Sigma(q))$ is the proposal best suited for sampling an energy function $U(q)$, and the local correlation $[\frac{\partial^2 U}{\partial q^2}]^{-1}$ could vary throughout the q -space. The dependence $\Sigma(q)$ results in additional computational complexity, because the corresponding Hamiltonian has the inseparable form $H(q, p) = U(q) + \frac{1}{2} (\log |\Sigma(q)| + p^\top \Sigma(q)^{-1} p)$, so $\frac{dq}{dt} = \frac{\partial H}{\partial p}$ and $\frac{dp}{dt} = -\frac{\partial H}{\partial q}$ are then functions of both q and p . In this case, Equations (9.4) through (9.6) are no longer a shear transformations and the Leapfrog Integrator no longer preserves volume. See the RMHMC section for more details.

9.4.2 The HMC Algorithm

There are three parts of an HMC step. Suppose the current state in the target space is q . First, a normal auxiliary variable p is sampled from $N(0, \Sigma)$. Then, L Leapfrog updates of step size ε are performed on the state (q, p) , and at the end of the trajectory p is negated to give a proposal state (q^*, p^*) . The negation of p is needed to ensure detailed balance but can be ignored if $K(p) = K(-p)$,

which is true when p is Gaussian. Finally, a Metropolis-Hastings step is needed to accept or reject the proposal (q^*, p^*) because the Leapfrog Integrator does not exactly preserve the Hamiltonian. After the acceptance step, p^* is discarded and a new p is generated for the next HMC step. In some HMC variants, p^* with a decay factor is used along with a freshly sampled p_i in the next step to encourage movement in the same direction between subsequent steps, but there is no evidence that this significantly improves sampling (see [148]). The standard HMC algorithm is given below.

HMC Algorithm

Input: Differentiable energy function $U(q)$, initial state $q_0 \in \mathbb{R}^n$, $n \times n$ p.d. covariance matrix Σ , step size ε , number of Leapfrog steps L , number of iterations N

Output: Markov Chain sample $\{q_1, \dots, q_N\}$ with stationary distribution U

For $i = 1 : N$,

1. Generate momentum $p_{i-1} \sim N(0, \Sigma)$.
2. Let $(q'_0, p'_0) = (q_{i-1}, p_{i-1})$. Perform L Leapfrog updates starting at (q'_0, p'_0) to reach a proposal state (q^*, p^*) as follows:

- (a) Do the first half-step update for p ,

$$p'_{\frac{1}{2}} = p'_0 - (\varepsilon/2) \frac{\partial U}{\partial q}(q'_0). \quad (9.12)$$

- (b) For $l = 1 : (L - 1)$, perform alternating full-step updates of q and p :

$$q'_l = q'_{l-1} + \varepsilon \Sigma^{-1} p'_{l-\frac{1}{2}}, \quad (9.13)$$

$$p'_{l+\frac{1}{2}} = p'_{l-\frac{1}{2}} - \varepsilon \frac{\partial U}{\partial q}(q'_l). \quad (9.14)$$

If $L = 1$, which is the LMC algorithm, skip this step.

- (c) Compute the final full-step q -update and the final half-step p -update

$$q'_L = q'_{L-1} + \varepsilon \Sigma^{-1} p'_{L-\frac{1}{2}}, \quad (9.15)$$

$$p'_L = p'_{L-\frac{1}{2}} - (\varepsilon/2) \frac{\partial U}{\partial q}(q'_L). \quad (9.16)$$

The proposed state is then $(q^*, p^*) = (q'_L, p'_L)$.

3. Accept the proposed state (q^*, p^*) according to the Metropolis-Hastings acceptance probability

$$\alpha = \min \left(1, \exp \left\{ - \left(U(q^*) + \frac{1}{2} (p^*)^\top \Sigma^{-1} p^* \right) + \left(U(q_{i-1}) + \frac{1}{2} p_{i-1}^\top \Sigma^{-1} p_{i-1} \right) \right\} \right). \quad (9.17)$$

If the proposal is accepted, then $q_i = q^*$. Otherwise, $q_i = q_{i-1}$. The momenta p_{i-1} and p^* can be discarded after the proposal.

Remark 1: To be fully correct, the proposed state at the end of Step 2 in the HMC algorithm should be $(q^*, p^*) = (q'_L, -p'_L)$ since a negation of the momentum at the end of the Leapfrog trajectory is

needed to ensure reversibility and detailed balance for HMC, as will be shown in the next section. Since $K(p) = K(-p)$ for Gaussian distributions, the calculation in Step 3 does not change whether $p^* = p'_L$ or $p^* = -p'_L$, and the negation can be safely ignored.

Remark 2: A different covariance matrix Σ can be used to generate each p_i . However, the same Σ must be used for the duration of a single proposal. Changing Σ between Leapfrog iterations breaks the shear structure of the Leapfrog updates, and detailed balance can no longer be guaranteed. This is a major obstacle for RMHMC approaches which take local manifold structure into account by allowing a dependence $\Sigma(q)$ based on the current location in the landscape.

The Metropolis-Hastings acceptance probability in Step 3 corresponds to a ratio of the joint density of $P(q, p)$:

$$\begin{aligned}\alpha &= \min\left(1, \frac{P(q^*, p^*)}{P(q_{i-1}, p_{i-1})}\right) \\ &= \min\left(1, \frac{\exp\{-H(q^*, p^*)\}}{\exp\{-H(q_{i-1}, p_{i-1})\}}\right) \\ &= \min\left(1, \frac{\exp\{-U(q^*) - K(p^*)\}}{\exp\{-U(q_{i-1}) - K(p_{i-1})\}}\right).\end{aligned}$$

The Leapfrog Update is deterministic, volume preserving, and exactly reversible, so no transition probabilities $K((q, p) \mapsto (q^*, p^*))$ appear in the Metropolis-Hastings ratio, only the density $P(q, p)$. The true continuous solution to Hamilton's Equations exactly satisfies $H(q, p) = H(q^*, p^*)$ for a proposal (q^*, p^*) generated according to Hamilton's Equations from an initial state (q, p) . Therefore, the Metropolis-Hastings acceptance probability would always be equal to 1 if the exact solution for Hamilton's Equation could be used.

However, since the Leapfrog discretization must be used instead, the value of H is not exactly conserved, and a Metropolis-Hastings step is needed to correct for this error. It is necessary to tune the sampling variables Σ , ε , and L correctly in order to obtain an accurate approximation to Hamilton's Equations and a high acceptance probability. In theory, HMC has a stationary distribution $\frac{1}{Z}e^{-U(q)}$ for any parameter setting, but good tuning is needed for good mixing as with any MCMC method. If the parameters are not tuned correctly, then either acceptance probabilities will be virtually 0 and the chain will never move, or the movements of the chain relative to the largest width of the local distribution will be so small that an unreasonably large number of updates must be used to achieve global movement.

9.4.3 The LMC Algorithm

Langevin Monte Carlo, or LMC, is simply the HMC algorithm where only $L = 1$ Leapfrog update is performed. LMC is equivalent to the Langevin Equation

$$q(t + \varepsilon) = q(t) - \frac{\varepsilon^2}{2} \Sigma^{-1} \frac{\partial U}{\partial q}(q(t)) + \varepsilon \sqrt{\Sigma^{-1}} z \quad (9.18)$$

(with $z \sim N(0, I_n)$) used in optimization with an additional p -update and Metropolis-Hastings acceptance step which results in a sampling algorithm on $\frac{1}{Z}e^{-U(q)}$. The LMC algorithm could be implemented using the HMC algorithm from the previous section with $L = 1$, but is usually done in a slightly more compact way with only one q -update and one p -update, as written below.

LMC Algorithm

Input: Differentiable energy function $U(q)$, initial state $q_0 \in \mathbb{R}^n$, $n \times n$ p.d. covariance matrix Σ , step size ε , number of iterations N

Output: Markov Chain sample $\{q_1, \dots, q_N\}$ with stationary distribution U

For $i = 1 : N$,

1. Generate momentum $p_{i-1} \sim \mathcal{N}(0, \Sigma)$.

2. Let $(q'_0, p'_0) = (q_{i-1}, p_{i-1})$. Update q according to the Langevin Equation:

$$q'_1 = q'_0 - \frac{\varepsilon^2}{2} \Sigma^{-1} \frac{\partial U}{\partial q}(q'_0) + \varepsilon \Sigma^{-1} p \quad (9.19)$$

and update p according to the Leapfrog Update

$$p'_1 = p'_0 - \frac{\varepsilon}{2} \frac{\partial U}{\partial q}(q'_0) + \frac{\varepsilon}{2} \frac{\partial U}{\partial q}(q'_1). \quad (9.20)$$

The proposed state is then $(q^*, p^*) = (q'_1, p'_1)$.

3. Accept the proposed state (q^*, p^*) according to the Metropolis-Hastings acceptance probability

$$\alpha = \min \left(1, \exp \left\{ - \left(U(q^*) + \frac{1}{2} (p^*)^\top \Sigma^{-1} p^* \right) + \left(U(q_{i-1}) + \frac{1}{2} p_{i-1}^\top \Sigma^{-1} p_{i-1} \right) \right\} \right). \quad (9.21)$$

If the proposal is accepted, then $q_i = q^*$. Otherwise, $q_i = q_{i-1}$. The momentum p_{i-1} can be discarded after the proposal.

The formulation of the q -update in (9.19) shows the two competing forces acting on the original space in LMC, and the same principles are work in each Leapfrog update of HMC. The term $-\frac{\varepsilon^2}{2} \Sigma^{-1} \frac{\partial U}{\partial q}(q'_0)$ is simply gradient descent rescaled by a p.d. matrix, corresponding roughly to "gravitational pull" in the energy landscape. When the momentum covariance is the Fisher Information $\Sigma(\theta) = \mathbb{E}_{X|\theta} \left[\frac{\partial^2 U}{\partial \theta^2}(X|\theta) \right]$ for a Bayesian inference problem given observations X , this term becomes the "natural gradient" $\Sigma^{-1}(\theta) \frac{\partial U}{\partial \theta}$ (Amari, [4]) which adapts to the local curvature of the parameter space. The Fisher Information is always positive-definite, and the natural gradient has better performance and invariance properties than the naive gradient $\frac{\partial U}{\partial \theta}$.

The term $\varepsilon \Sigma^{-1} p = \varepsilon \sqrt{\Sigma^{-1}} z$ for $z \sim \mathcal{N}(0, I_n)$ corresponds roughly to random "winds". The gravitational pull of the gradient term should overpower the random forces of the diffusion term when moving down an energy slope, but the diffusion term becomes dominant once a local minimum has been reached and only movement along level curves is possible. An informed choice of Σ is needed to ensure that the random diffusion forces can make meaningful proposals once the chain has reach the bottom of an energy basin. If $\Sigma(q) \approx \frac{\partial^2 U}{\partial q^2}$, then $\sqrt{\Sigma(q)^{-1}} z \sim \mathcal{N}(0, \Sigma(q)^{-1})$ and the diffusion forces follow the local covariance structure $[\frac{\partial^2 U}{\partial q^2}]^{-1}$, so that the "wind" primarily blows along the local manifold. Imagine that the local landscape is a canyon. If the winds blow perpendicular to the direct of the canyon, the steep edges of the canyon will prevent any meaningful movement. However, if the wind happens to blow parallel to the canyon, then movement through the canyon becomes possible.

LMC has different properties than "full" HMC where a substantial number of Leapfrog updates

L are used. Since the momenta are discarded after only a single step in LMC, successive proposals are not encouraged to move in the same direction and LMC explores the landscape $U(q)$ in a random walk. The gradient information used in the LMC update allows some features of the energy landscape to be taken into account unlike Random-Walk Metropolis, but HMC has better theoretical scaling properties because repeatedly updating the same momentum p can lead to longer trajectories in the joint space.

However, LMC can be more useful in some situations. It is more practical and accurate to implement an approximation of the dynamics of a q -dependent momentum $p \sim N(0, \Sigma(q))$ for LMC than HMC, as will be discussed in the RMHMC section. In complex landscapes, the benefits of HMC are limited by the instability of the Leapfrog dynamics, and often the number of Leapfrog steps L must be kept small to achieve reasonable acceptance rates, in which case HMC and LMC are very similar.

9.4.4 Proof of Detailed Balance for HMC

The simplest way to show that an MCMC sampling method preserves a distribution P is to show that the method satisfies the detailed balance relation

$$P(x)T(x \mapsto x^*) = P(x^*)T(x^* \mapsto x) \quad (9.22)$$

for the proposal density T defined by the MCMC method. A proof that HMC satisfies detailed balance is given below. It can also be shown that HMC is ergodic and guaranteed to explore the entire q -space, provided that ε is selected from a small random interval at the beginning of each HMC update. This random selection of ε is needed to ensure ergodicity in theory, because there is the possibility for exactly or nearly periodic orbits to occur during HMC, but this phenomenon only exists for a narrow band of ε . See [128] for more details.

Theorem 9.1. *The HMC Algorithm satisfies detailed balance and has a stationary distribution $P(q) = \frac{1}{Z}e^{-U(q)}$.*

Proof. It is sufficient to show that an HMC step satisfies detailed balance for the joint distribution $P(q, p) = \frac{1}{Z}e^{-U(q)-K(p)}$ in order to show that the stationary distribution of q in the HMC process follows $\frac{1}{Z}e^{-U(q)}$, as shown in Equation (1.11). For this proof, $p \sim \frac{1}{Z}e^{-K(p)}$ for a smooth energy function K on \mathbb{R}^n , which might not be Gaussian or symmetric. In practice $p \sim N(0, \Sigma)$ and $K(p) = (1/2)p^\top \Sigma^{-1}p$.

Let $q \sim \frac{1}{Z}e^{-U(q)}$. After generating $p \sim \frac{1}{Z}e^{-K(p)}$ in Step 1 in the HMC algorithm, it is clear that $(q, p) \sim P(q, p)$ because of the independence of q and p implied by the factorizable form of the joint density. Let the proposal (q^*, p^*) be the state reached after performing L Leapfrog steps of size ε from the state (q, p) and negating p at the end of the trajectory. As shown in Section 9.3.3, each Leapfrog step is a change of coordinates with determinant 1, and the negation of p at the end of the trajectory is a change of coordinates with Jacobian determinant of absolute value 1. Therefore $(q, p) \mapsto (q^*, p^*)$ is a change of coordinates with a Jacobian determinant of absolute value 1, since the determinant of the composition of coordinate changes is the product of the determinants of each change. By the change of coordinate rule for probability densities

$$g(y) = f(x) \left| \det \left(\frac{dx}{dy} \right) \right|$$

where $f(x)$ is the original density and $g(y)$ is the new density for the mapping $x \mapsto y$, it follows that (q^*, p^*) has the same density function as (q, p) because $|\det(dx/dy)| = 1$. The proposal is also exactly reversible, since applying L Leapfrog steps of size ε to (q^*, p^*) and negating p^* at the end of the trajectory will give the original state (q, p) .

Since the mapping $(q, p) \mapsto (q^*, p^*)$ is deterministic and reversible, the proposal density T defined by the HMC Algorithm is

$$T((q, p) \mapsto (q^*, p^*)) = \min(1, \exp\{-(U(q^*) + K(p^*)) + (U(q) + K(p))\}),$$

$$T((q, p) \mapsto (q', p')) = 0 \quad \text{if} \quad (q', p') \neq (q^*, p^*).$$

Similarly, the transition density starting from (q^*, p^*) is non-zero only for the proposal (q, p) and has the form

$$T((q^*, p^*) \mapsto (q, p)) = \min(1, \exp\{-(U(q) + K(p)) + (U(q^*) + K(p^*))\}).$$

The detailed balance equation (9.22) for HMC is

$$\frac{1}{Z} e^{-U(q) - K(p)} \min\left(1, \frac{\exp\{-U(q^*) - K(p^*)\}}{\exp\{-U(q) - K(p)\}}\right) = \frac{1}{Z} e^{-U(q^*) - K(p^*)} \min\left(1, \frac{\exp\{-U(q) - K(p)\}}{\exp\{-U(q^*) - K(p^*)\}}\right)$$

which is clearly true. Therefore HMC satisfies detailed balance and preserves the joint distribution $\frac{1}{Z} e^{-U(q) - K(p)}$. \square

9.4.5 Tuning Standard HMC

This section discusses tuning the ε and L parameters in the standard HMC setting with a fixed Σ . It is sufficient to only consider $\Sigma = I_n$, because the results can naturally be extended to an arbitrary Σ by rescaling the q -space as in Lemma 9.1. Tuning Σ itself is the main topic of the RMHMC section.

The step size ε must be small enough for the Leapfrog Integrator to accurately simulate Hamilton's Equations in order for H to stay approximately constant throughout the Leapfrog updates. Since the Metropolis-Hastings acceptance depends on difference in H between the original state and the proposal, smaller step sizes tend to have higher acceptance rates because H will not change as much. On the other hand, if ε is too small then the chain will remain nearly stationary and effective sampling becomes impossible. A simple way to tune HMC in a new environment is to set $\Sigma = I_n$, $L = 1$, and vary ε until acceptance rates of 40% – 85% are obtained. This range provides a good balance between high acceptance and good movement, and more extreme ε in either direction are unlikely to substantially improve HMC performance. Different values of ε might be needed in different areas of the state space.

When $\Sigma = I_n$, the ideal step size ε^* should roughly equal the width of $U(q)$ in the most constrained linear direction of the local region of the energy landscape. If the landscape is Gaussian or approximately Gaussian, ε^* should be close to the square root of the smallest eigenvalue of the local covariance matrix. The q -update in (9.13) will lead to low acceptance rates when ε is substantially larger than the smallest marginal standard deviation of $U(q)$ because the spherical auxiliary variable will make unlikely proposals along the most constrained direction. On the other hand, proposals should be accepted with reasonably high probability when ε and the smallest standard deviation are about equal because local deviations in any direction will give states with about the same energy as the current state.

Since each Leapfrog update moves a distance of about ε in the q -space, ignoring the effect of the gradient, and ε is limited to at most the smallest marginal standard deviation of q , the number of leapfrog steps L^* needed to reach a nearly independent state in a single HMC step is $L^* \approx \sqrt{\lambda_{\max}}/\varepsilon^*$, where λ_{\max} is the largest eigenvalue of the local covariance of the q -space. Remember that the Hamiltonian trajectory is not a random walk and tends to move in the same direction unless obstacles in the energy landscape are encountered, so displacement scales linearly with the number of steps L . In simple landscapes, L can be quite large (over 100) because the Leapfrog dynamics are very accurate, but in more complex landscapes the acceptance rates for HMC trajectories can drop very quickly when L becomes too large. If εL is not on the order of the largest standard deviation, HMC will exhibit strong autocorrelation and will not be able to move through the space effectively. See the HMC in Practice section for experimental demonstrations of these principles.

The local correlation could vary drastically throughout the state space, and a parameter setting that is effective in one mode of the landscape might perform badly in another mode. However, using RMHMC alleviates these problems because ε becomes a "dimensionless" quantity and small numbers of Leapfrog steps (even $L = 1$) can still provide good movement through the space.

9.5 Riemann Manifold HMC

Riemann Manifold HMC (or RMHMC) extends the standard HMC method by allowing the covariance matrix for the auxiliary momentum variables p to have a dependency $\Sigma(q)$ on the current location q in the energy landscape. This can vastly improve the sampling properties of HMC, especially in situations where the distribution of q is concentrated along a low-dimensional manifold in the state space.

Traditional HMC with $\Sigma = I_n$ is ineffective in these situations because the step size needed for acceptance must be on the order of the smallest standard deviation of q , which will be orders of magnitude smaller than the standard deviations along the primary manifold dimensions. Using a large number of Leapfrog steps L can only partially compensate for this discrepancy, and in complex landscapes the trajectory can become unstable when L is too large.

RMHMC, on the other hand, uses local geometry to make proposals in meaningful directions along the local manifold, leading to better sampling with only a few Leapfrog steps. The dependency $\Sigma(q)$ complicates the dynamics of RMHMC and requires additional computational considerations, some of which are very problematic. While an exact RMHMC implementation is infeasible in many practical situations, approximate implementations could provide many of the benefits of RMHMC in a flexible and generalizable framework.

9.5.1 Linear Transformations in HMC

The lemma below states an important invariance property of HMC under a certain type of linear transformation which provides insight into the effect of the momentum covariance Σ in HMC dynamics.

Lemma 9.1. Let $U(q)$ be a smooth energy function and let $p \sim N(0, \Sigma)$ be the distribution of the HMC auxiliary variables for p.d. matrix Σ , and let A be an invertible matrix. The HMC dynamics of (q, p) initialized at (q_0, p_0) are equivalent to the HMC dynamics of $(q', p') = (Aq, (A^\top)^{-1}p)$ initialized at $(q'_0, p'_0) = (Aq_0, (A^\top)^{-1}p_0)$ because $(Aq_t, (A^\top)^{-1}p_t) = (q'_t, p'_t)$ for any Leapfrog step $t \geq 1$.

Proof. Let $(q', p') = (Aq, (A^\top)^{-1}p)$. By the change of variables formula for probability densities, $P'(q') = P(q)/|\det(A)|$, and since A is constant, the new denominator is absorbed into the normalizing constant, so the energy functions of q and q' differ only by an additive constant: $U'(q') = U(A^{-1}q') + c$. Using the chain rule,

$$\frac{\partial U'}{\partial q'}(q^*) = (A^\top)^{-1} \frac{\partial U}{\partial q}(A^{-1}q^*)$$

for any vector q^* . The transformed momenta have a distribution $p' \sim N(0, (A^\top)^{-1}\Sigma A^{-1})$ and energy function $K'(p') = A\Sigma^{-1}A^\top p'$. One Leapfrog update of $(q'_0, p'_0) = (Aq_0, (A^\top)^{-1}p_0)$ is given by

$$p'_{1/2} = p'_0 - \frac{\varepsilon}{2} \frac{\partial U'}{\partial q'}(q'_0) = (A^\top)^{-1}p_0 - \frac{\varepsilon}{2} (A^\top)^{-1} \frac{\partial U}{\partial q}(q_0) \quad (9.23)$$

$$q'_1 = q'_0 + \varepsilon A \Sigma^{-1} A^\top p'_{1/2} = Aq_0 - \frac{\varepsilon^2}{2} A \Sigma^{-1} \frac{\partial U}{\partial q}(q_0) + \varepsilon A \Sigma^{-1} p_0 \quad (9.24)$$

$$p'_1 = p'_{1/2} - \frac{\varepsilon}{2} \frac{\partial U'}{\partial q'}(q'_1) = (A^\top)^{-1}p_0 - \frac{\varepsilon}{2} (A^\top)^{-1} \frac{\partial U}{\partial q}(q_0) - \frac{\varepsilon}{2} (A^\top)^{-1} \frac{\partial U}{\partial q}(A^{-1}q'_1). \quad (9.25)$$

Multiplying (9.23) and (9.25) by A^\top and multiplying (9.24) by A^{-1} gives the Leapfrog update for the original pair (q_0, p_0) , and it is clear that $(q'_1, p'_1) = (Aq_1, (A^\top)^{-1}p_1)$. By induction, this relation must hold for any number of Leapfrog steps. \square

Remark: In practice, the equivalence in Lemma 9.1 is not exact because of computational inaccuracies arising from the matrix operations performed with A . However, if A is well-conditioned, numeric implementations of the two chains should give very similar results.

This lemma provides a key insight about tuning HMC. Suppose that the distribution of q is approximately Gaussian in some region around a point q^* with p.d. covariance Σ_{q^*} . The covariance can be decomposed as $\Sigma_{q^*} = AA^\top$ by several possible methods such as Cholesky or eigenvalue decomposition. Consider the HMC dynamics of the chain (q, p) with $p \sim N(0, \Sigma_{q^*}^{-1})$. By Lemma 9.1, the dynamics of (q, p) are equivalent to the dynamics of $(A^{-1}q, A^\top p)$. Now, $A^\top p \sim N(0, I_n)$, and $\text{Var}(A^{-1}q) = I_n$ in a region near q^* , so the transformed position and momentum variables are approximately independent with variance 1 in each dimension. Since the transformed space is easy to sample, the HMC dynamics of $(A^{-1}q, A^\top p)$ should lead to nearly independent states in a small number number of Leapfrog steps (even $L = 1$), and the same sampling properties hold for the equivalent dynamics of the original (q, p) .

This observation is one of several motivations for using local curvature information to improve the performance of HMC dynamics, as is done in RMHMC. Let q^* be a position in the energy landscape, and suppose $\frac{\partial U^2}{\partial^2 q}(q^*)$ is positive definite, so that $\Sigma_{q^*} = [\frac{\partial U^2}{\partial^2 q}(q^*)]^{-1}$ gives the local correlation and scaling structure of U in a neighborhood around q^* . In general, $\frac{\partial U^2}{\partial^2 q}(q^*)$ may not be positive definite, but a p.d. relative Σ'_{q^*} obtained by thresholding the eigenvalues of $\frac{\partial U^2}{\partial^2 q}(q^*)$ and inverting could provide the same benefits.

By the discussion above, using the momentum $p \sim N(0, \Sigma_{q^*}^{-1})$ should lead to a nearly independent proposal in the q -space in a small number of Leapfrog steps, so $\Sigma_{q^*}^{-1}$ is the ideal proposal covariance at the point q^* . As discussed in Section 9.4.3, using $\Sigma_{q^*}^{-1}$ as the covariance for the momenta promotes movement along the local manifold and allows the chain to travel along level curves near bottom of energy basins, which is impossible using gradient information alone. In order for HMC to be an effective sampling method, instead of just an optimization method, it is necessary to use an informative momentum covariance. If estimates s_i of the marginal standard deviation can be obtained, then a diagonal covariance Λ where $\lambda_i = 1/s_i^2$ can account for differences in scale between the variables. However, the s_i might vary throughout the state space, and a diagonal covariance Λ cannot account for correlations between the dimensions, which are usually strong in real world problems.

The following lemma gives three equivalent ways of implementing the HMC dynamics of a chain (q, Cp) for an invertible matrix C , and where the original momentum distribution is $p \sim N(0, \Sigma)$. In RMHMC, $\Sigma = I_n$ and $C = \sqrt{\partial U^2 / \partial^2 q}$, assuming that the curvature is positive definite. Although equivalent, the computational cost of these implementations can vary depending on the matrix decompositions and inversions required. When working in high dimensions with large matrices, the cost of matrix operations quickly balloons, and care is needed to make sure that the chain can be updated in a reasonable amount of time.

Lemma 9.2. Let $U(q)$ be a smooth energy function, Σ a p.d. matrix, and C an invertible matrix. The dynamics of the following HMC chains are equivalent:

1. The momentum is sampled from $p \sim N(0, C^\top \Sigma C)$ and the chain is updated according to the standard HMC dynamics of (q, p) .
2. The momentum is sampled from $p \sim N(0, \Sigma)$ and the chain is updated according to the standard HMC dynamics of $(q, C^\top p)$, i.e.
3. The momentum is sampled from $p \sim N(0, \Sigma)$ and the chain is updated according to the altered HMC dynamics defined by

$$\frac{dq}{dt} = C^{-1} \frac{\partial K}{\partial p}, \quad (9.26)$$

$$\frac{dp}{dt} = -[C^{-1}]^\top \frac{\partial U}{\partial q}. \quad (9.27)$$

Moreover, (2) and (3) can both be implemented using the altered Leapfrog update

$$p_{t+1/2} = p_t - \frac{\varepsilon}{2}[C^{-1}]^\top \frac{\partial U}{\partial q}(q_t) \quad (9.28)$$

$$q_{t+1} = q_t + \varepsilon C^{-1} \Sigma^{-1} p_{t+1/2} \quad (9.29)$$

$$p_{t+1} = p_{t+1/2} - \frac{\varepsilon}{2}[C^{-1}]^\top \frac{\partial U}{\partial q}(q_{t+1}). \quad (9.30)$$

Proof. Consider the dynamics of (2). Sample $p_0 \sim N(0, \Sigma)$, and let $p'_0 = C^\top p_0$, which means p'_0 is distributed $N(0, C^\top \Sigma C)$. The Leapfrog updates of $(q, p') = (q, C^\top p)$ are given by

$$p'_{t+1/2} = p'_t - \frac{\varepsilon}{2} \frac{\partial U}{\partial q}(q_t) \quad (9.31)$$

$$q_{t+1} = q_t + \varepsilon C^{-1} \Sigma^{-1} [C^{-1}]^\top p'_{t+1/2} \quad (9.32)$$

$$p'_{t+1} = p'_{t+1/2} - \frac{\varepsilon}{2} \frac{\partial U}{\partial q}(q_{t+1}) \quad (9.33)$$

which are identical to the standard Leapfrog updates of (1), proving the equivalence between (1) and (2). On the other hand, multiplying (9.31) and (9.33) by $[C^{-1}]^\top$ gives the same updates as (9.28) through (9.30), because $p_t = [C^{-1}]^\top p'_t$ at each step t . The updates (9.28) through (9.30) are easily identified as the Leapfrog dynamics of the altered Hamiltonian Equations (9.26) and (9.27), showing the equivalence of (2) and (3). \square

The above lemma is important for two reasons. First, it shows that HMC dynamics with $p \sim N(0, M)$ with $M = C^\top C$ can be interpreted as the HMC dynamics resulting from a momentum $p \sim N(0, I_n)$ with the altered form of Hamilton's Equations in (9.26) and (9.27). This provides an important link between RMHMC and other "skew-symmetric" HMC methods that alter Hamilton's Equations in a similar way, most importantly stochastic gradient HMC.

Second, the lemma provides an alternative way to implement the dynamics of $p \sim N(0, M)$ that only requires the calculation of $\sqrt{M^{-1}}$, not M itself. This follows from letting $\Sigma = I_n$, $C = \sqrt{M}$, and observing that the updates in (9.28) through (9.30) require only C^{-1} . The ideal momentum covariance is $\frac{\partial U^2}{\partial^2 q}$, and in convex regions $\sqrt{[\frac{\partial U^2}{\partial^2 q}]^{-1}}$ can be approximated from a sample of local positions using a variant of the LBFGS algorithm. The calculation requires no matrix inversion or decomposition, and provides a computationally efficient way to implement an approximate RMHMC algorithm, as is discussed later. Still, obtaining an accurate estimate of the root inverse Hessian in complex landscapes is a substantial obstacle for any RMHMC implementation.

9.5.2 RMHMC Dynamics

Linear transformation to adapt to local curvature has clear theoretical benefits, and allows for nearly independent sampling in only a few Leapfrog steps. However, HMC dynamics that include local curvature information are much more difficult to discretize than standard HMC dynamics, and costly computational methods are needed.

In standard HMC, the same matrix Σ is used as the covariance of the momenta throughout a single proposal. In RMHMC, there is a dependence of the momentum covariance $\Sigma(q)$ on the current position q in the energy landscape. For now, $\Sigma(q)$ is any smooth matrix function of q that returns a p.d. symmetric matrix, but in practice this matrix should reflect the local curvature near

a position q in the energy landscape. A discussion of the choice of $\Sigma(q)$ in practice can be found later in this section. The RMHMC momentum distribution is $p \sim N(0, \Sigma(q))$, with energy function

$$K(q, p) = \frac{1}{2} \log((2\pi)^n |\Sigma(q)|) + \frac{1}{2} p^\top \Sigma(q)^{-1} p, \quad (9.34)$$

and the joint Hamiltonian is

$$H(q, p) = U(q) + \frac{1}{2} \log((2\pi)^n |\Sigma(q)|) + \frac{1}{2} p^\top \Sigma(q)^{-1} p. \quad (9.35)$$

The momentum energy function must include an extra term $\frac{1}{2} \log((2\pi)^n |\Sigma(q)|)$ not found in standard HMC, and evaluating derivatives of this term is a source of computational difficulty. Observe that

$$\int_{\mathbb{R}^n} \frac{1}{Z} e^{-H(q, p)} dp = \frac{1}{Z} e^{-U(q)} \int_{\mathbb{R}^n} \frac{1}{\sqrt{(2\pi)^n |\Sigma(q)|}} e^{-p^\top \Sigma(q)^{-1} p} dp = \frac{1}{Z} e^{-U(q)}, \quad (9.36)$$

so the marginal distribution of q is the target distribution, and the q -sample obtained from updating (q, p) jointly will follow the correct distribution, just as in standard HMC. The Hamiltonian Equations governing RMHMC are

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = \Sigma(q)^{-1} p, \quad (9.37)$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q} = -\frac{\partial U}{\partial q} - \frac{1}{2} \text{Tr} \left[\Sigma(q)^{-1} \frac{\partial \Sigma(q)}{\partial q} \right] + \frac{1}{2} p^\top \Sigma(q)^{-1} \frac{\partial \Sigma(q)}{\partial q} \Sigma(q)^{-1} p. \quad (9.38)$$

The updates of q and p are no longer separable, since the update of p depends on both q and the current p . Therefore, if the Leapfrog integrator were used, the coordinate changes would no longer have a shear structure, so there is no guarantee that the Leapfrog coordinate change has determinant 1 for RMHMC dynamics. This upsets the detailed balance of HMC, and naively implementing the Leapfrog integrator in an RMHMC setting does not preserve the distribution of $\frac{1}{Z} e^{-U(q)}$.

To overcome the non-separability of the RMHMC update equations, the update values are defined by an implicit set of equations that must be solved using fixed point iteration. One iteration of the Generalized Leapfrog Integrator for discretizing the dynamics of a non-separable joint Hamiltonian H is given by

$$p_{t+1/2} = p_t - \frac{\varepsilon}{2} \frac{\partial H}{\partial q}(q_t, p_{t+1/2}), \quad (9.39)$$

$$q_{t+1/2} = q_t + \frac{\varepsilon}{2} \left[\frac{\partial H}{\partial p}(q_t, p_{t+1/2}) + \frac{\partial H}{\partial p}(q_{t+1}, p_{t+1/2}) \right], \quad (9.40)$$

$$p_{t+1} = p_{t+1/2} - \frac{\varepsilon}{2} \frac{\partial H}{\partial q}(q_t, p_{t+1/2}). \quad (9.41)$$

The update in the first two steps is implicitly defined, allowing for the simulation of the dynamics of a non-separable H . In the case of standard HMC, $H(q, p) = U(q) + K(p)$ and the Generalized Leapfrog updates are identical to the standard Leapfrog scheme. When H is non-separable, fixed point iteration must be used to solve (9.39) and (9.40). The details of the fixed point updates are included with the RMHMC Algorithm later in the section.

It can be shown that the Generalized Leapfrog updates are volume-preserving, and that RMHMC maintains detailed balance and preserves the target distribution. The proof is similar to the proof of detailed balance for standard HMC, with appropriate adjustments made for the proofs of volume preservation and reversibility based on the Generalized Leapfrog updates. However, RMHMC is not explicitly reversible like standard HMC, because solutions to the fixed point equations will not be exact in practice and reversing the fixed point calculations will not give the exact original position.

9.5.3 RMHMC Algorithm and Variants

There are several variants of the RMHMC algorithm. First the full RMHMC algorithm is presented, which requires fixed point iteration and the calculation of the derivative of $\Sigma(q)$. Since $\Sigma(q)$ in practice is the local curvature, full RMHMC requires calculation of the third derivatives of the target energy U , which is a large computational burden and is impossible in many practical situations. There is an RMHMC variant for $L = 1$ Leapfrog updates which requires the calculations of third derivatives of U at the original and proposed state but involves no fixed point iteration. The details of this algorithm are slightly different than full RMHMC, and the interested reader should refer to [85].

Full RMHMC Algorithm

Input: Differentiable energy function $U(q)$, initial state $q_0 \in \mathbb{R}^n$, $n \times n$ differentiable p.d. covariance function $\Sigma(q)$, step size ε , number of iterations N , number of Leapfrog steps L , number of fixed point steps K

Output: Markov Chain sample $\{q_1, \dots, q_N\}$ with stationary distribution U

For $i = 1 : N$,

1. Generate momentum $p_{i-1} \sim N(0, \Sigma(q_{i-1}))$.
2. Let $(q'_0, p'_0) = (q_{i-1}, p_{i-1})$. For $l = 1 : L$, update (q'_{l-1}, p'_{l-1}) using the Generalized Leapfrog integrator to reach a proposal state $(q^*, p^*) = (q'_L, p'_L)$ as follows:

- (a) Let $\hat{p}_0 = p'_{l-1}$. For $k = 1 : K$, update \hat{p}_{k-1} according to the fixed point equation

$$\hat{p}_k = \hat{p}_{k-1} - \frac{\varepsilon}{2} \Sigma(q'_{l-1})^{-1} \hat{p}_{k-1} \quad (9.42)$$

to obtain the half-step momentum update $p'_{l-1/2} = \hat{p}_K$.

- (b) Let $\hat{q}_0 = q'_{l-1}$. For $k = 1 : K$, update \hat{q}_{k-1} according to the fixed point equation

$$\hat{q}_k = \hat{q}_{k-1} - \frac{\varepsilon}{2} \Sigma(q'_{l-1})^{-1} \hat{q}_{k-1} \quad (9.43)$$

where $\partial H / \partial p$ is given in (9.38), to obtain the full step position update $q'_l = \hat{q}_K$.

- (c) Update $p'_{l-1/2}$ according to the explicit equation

$$p'_l = p'_{l-1/2} - \frac{\varepsilon}{2} \Sigma(q'_l)^{-1} p'_{l-1/2} \quad (9.44)$$

to obtain the full step momentum update p'_l .

3. Accept the proposed state (q^*, p^*) according the Metropolis-Hastings acceptance probability

$$\alpha = \min(1, \exp\{-H(q^*, p^*) + H(q_{i-1}, p_{i-1})\}) \quad (9.45)$$

where $H(q, p)$ is the joint Hamiltonian of (9.35). If the proposal is accepted, then $q_i = q^*$. Otherwise, $q_i = q_{i-1}$. The momentum p_{i-1} can be discarded after the proposal.

Remark: The step size ε is a "dimensionless" quantity, because the RMHMC dynamics should locally correspond to the trivial HMC distribution where both q and p are $N(0, I_n)$. The scale of RMHMC is implicitly the standard normal scale, and therefore setting ε to a value slightly less than 1, the smallest (and largest) rescaled standard deviation, should yield good results for any RMHMC algorithm.

To alleviate the difficulties of full RMHMC, one can use an approximate RMHMC algorithm where the covariance $\Sigma(q_{t-J+1}, q_{t-J+2}, \dots, q_t)$ can depend on the previous states of the chain before the momentum is sampled, but is fixed throughout the Leapfrog updates. This variant is essentially the standard HMC algorithm with a principled way of changing Σ between proposals.

Changing Σ between proposals does not violate the preservation of the target distribution, because if q has the correct distribution after an update with covariance Σ_0 , q will still follow the correct distribution after an HMC update with any covariance Σ_1 , not necessarily equal to Σ_0 . At first it might appear that using the previous states to obtain $\Sigma(q_{t-J+1}, q_{t-J+2}, \dots, q_t)$ might violate the Markovian structure of HMC, but this is not the case, because detailed balance holds for any $\Sigma(x_1, \dots, x_J)$, and in particular it is not required that (x_1, \dots, x_J) have the target distribution. Therefore there is no distributional dependence except of the current state q_t , and the same proofs of detailed balance still hold.

Although the simplified RMHMC algorithm cannot capture the full dependence implied by RMHMC dynamics, since $\Sigma(q)$ is not updated through the Leapfrog iterations, it is computationally identical to standard HMC and provides an efficient way to incorporate curvature information by using a quasi-Newton estimate of the inverse Hessian. Even if this information is approximate, it can still substantially improve the movement of the chain in the energy landscape.

Simplified RMHMC Algorithm

Input: Differentiable energy function $U(q)$, initial state $q_0 \in \mathbb{R}^n$, $n \times n$ p.d. covariance function $\Sigma(q_1, \dots, q_J)$, step size ε , number of iterations N , number of previous states in memory J

Output: Markov Chain sample $\{q_1, \dots, q_N\}$ with stationary distribution U

For $i = 1 : N$,

1. Calculate the current covariance matrix $\Sigma^* = \Sigma(q_{i-1-J}, q_{i-J}, \dots, q_{i-1})$.
2. Update (q_{i-1}, p_{i-1}) according to standard HMC dynamics using the proposal covariance Σ^* .

Remark: Since Σ^* is fixed throughout updates, only a small number of Leapfrog steps (usually $L = 1$) are used in the simplified RMHMC algorithm, because the local curvature varies with each update.

9.5.4 Covariance Functions in RMHMC

The RMHMC algorithm and its simplification preserve the target for any differentiable p.d. covariance function $\Sigma(q)$, but to actually improve sampling $\Sigma(q)$ must reflect the local curvature of the space. In general, $\partial U / \partial q$ is not necessarily p.d., so the naive choice $\Sigma(q) = \frac{\partial U}{\partial q}(q)$ is not always possible in practice.

The original authors of RMHMC restrict their attention to sampling from a posterior probability $p(\theta|X)$ for a family of probability models $p(X|\theta)$. In this case, there is a natural choice for the proposal covariance given by the Fisher Information

$$\Sigma(\theta) = -E_{X|\theta} \left[\frac{\partial^2}{\partial \theta^2} \log p(X|\theta) \right] \quad (9.46)$$

which is guaranteed to be positive definite. In simple cases the Fisher Information can be obtained analytically. If this is not possible, it can be estimated by taking the expectation of the curvature

over the observed data and thresholding the eigenvalues of the resulting matrix for stability. The positive definite structure of the Fisher Information is a nice property in theory but in practice the matrix must be estimated and very small or even negative eigenvalues can still be encountered. When the Fisher Information is used, the gradient term $\Sigma(\theta)^{-1} \frac{\partial U}{\partial \theta}(\theta)$ that appears in the Langevin equation corresponds to the natural gradient of Amari et al [4].

The Fisher Information is not a complete solution because it can only be used when sampling a parameter θ for a family of distributions with a set of observed data X . When sampling from just a probability distribution $P(q) = \frac{1}{Z} e^{-U(q)}$, there is no way to make the curvature p.d. by taking an expectation. However, the largest eigenvalues of $\frac{\partial U}{\partial q}$ should be the most important for the dynamics of HMC. This is because of largest eigenvalues of the curvature represent the most constrained linear dimensions of the distribution. When near a local minima, negative eigenvalues or eigenvalues near 0 are not problematic, because movement in these directions leave H approximately constant or decrease H . Thresholding the eigenvalues of $\frac{\partial U}{\partial q}$ can give a p.d. covariance that preserves the most important local geometric information when the curvature itself is not p.d.

Another option is to estimate the local curvature $\Sigma(q)$ as is done in quasi-Newton methods. This type of method of method uses a sequence of past states $q_{t+1-J}, q_{t+2-J}, \dots, q_t$ to estimate the inverse Hessian at the current state q_t . As demonstrated in Lemma 9.2, only the root inverse Hessian is needed to simulate the HMC dynamics of $\Sigma(q) = \frac{\partial U}{\partial q}$, and there is a variant of the LBFGS algorithm that estimates the root inverse Hessian directly. See [26] and [215] for details. Because of the difficulty of estimating matrices, some adjustment of the estimated eigenvalues might be necessary anyway, and an SVD decomposition may be necessary for good results. If this is the case, it is simpler to use the ordinary LBFGS algorithm and find the root inverse by taking the square root of the decomposed eigenvalues after adjustment. See Section 9.6.1 for an example of this in a simulated experiment.

9.6 HMC in Practice

In this section, three applications of HMC and related algorithms are presented. The first application is a toy experiment using Gaussian distributions that are highly constrained in all but a few directions. This experiment is very useful for understanding the basic principles behind tuning the HMC parameters. Next, sampling the parameters of a logistic regression model is considered. This setting is one of the few scenarios where the full RMHMC algorithm can be implemented in practice because the Fisher Information is available in closed form, and it allows for direct comparison between the different HMC models. Finally, the Alternating Backwards Propagation algorithm is presented, which uses HMC as a key step when sampling from distributions defined over high-dimensional parameters and images.

9.6.1 Simulated Experiments on Constrained Normal Distributions

In this section, HMC and variants are used to sample from normal distributions that are highly constrained in all but a few directions. Such distributions are challenging for any MCMC method that uses local updates based on the current position in the energy landscape, because it is difficult to efficiently sample the unconstrained dimensions while still remaining in the tightly constrained region of the landscape.

Two different distributions are considered: $N(0, \Sigma_1)$ and $N(0, \Sigma_2)$. Both Σ_1 and Σ_2 are 100×100 diagonal matrices. The first 15 entries of Σ_1 and Σ_2 are 1, representing the unconstrained sampling direction. The last 85 entries of Σ_1 are 0.01^2 , and the last 85 entries of Σ_2 are 0.0001^2 . Σ_1 is an easier sampling case, because the ratio between the largest and smallest standard deviations is around 100, so about 100 Leapfrog steps should be needed for effective sampling with standard HMC. Σ_2 is a more difficult scenario, because the ratio between the largest and smallest standard deviations is 10,000. When the local covariance exhibits such an extreme difference in scales, HMC is no longer an effective sampling method because the Leapfrog approximation becomes very unstable when very

large number of steps are used, as can be seen from the Australian Credit Data in Section 9.6.2. This can be addressed by including approximate curvature information with a quasi-Newton HMC variant. All experiments use 5000 burn-in iterations and 10,000 sampling iterations.

The energy functions of the two target distributions have the form $U_1(q) = \frac{1}{2}q^\top \Sigma_1^{-1}p$ and $U_2 = \frac{1}{2}q^\top \Sigma_2^{-1}q$. Like all normal distributions, the target distributions have constant curvature Σ_1^{-1} and Σ_2^{-1} respectively. The full RMHMC algorithm can be implemented by simply letting $p \sim N(0, \Sigma_1^{-1})$ or $p \sim N(0, \Sigma_2^{-1})$ for each update, and the standard dynamics can be used because the curvature $\Sigma(q)$ is constant throughout the state space, so the derivative of $\Sigma(q)$ is 0 and the Generalized Leapfrog update becomes identical to the standard Leapfrog update. In this experiment, the RMHMC algorithm with only one Leapfrog update can obtain virtually independent samples from the target distribution in every iteration.

Consider the target distribution $N(0, \Sigma_1)$. Three different methods are used to sample this distribution: Random-Walk Metropolis Hastings, LMC, and HMC with $L = 150$ Leapfrog updates. The momentum covariance was set to I_{100} for both HMC and LMC. A step size of 0.04 was used for RW Metropolis, and a step size of 0.008 was used for LMC and HMC, since 0.008 is slightly less than the smallest marginal standard deviation of 0.01. The first two methods cannot effectively sample the target distribution, because both methods will be forced to explore the unconstrained directions of the distribution with a random walk with a small step size. LMC has difficulty in this situation because the momenta are refreshed after a single update, whereas HMC uses the same momentum for a large number of updates, so HMC does not explore the distribution via random-walk like the other two methods. Since $\varepsilon L = 1.5$ for HMC, and the largest marginal standard deviation of the target distribution is 1, HMC obtains nearly independent samples in every iteration. For fair comparison between the methods, 150 RW Metropolis and LMC updates are counted as a single iteration in the figures below.

Next, consider the target distribution $N(0, \Sigma_2)$. The ratio between the largest and smallest standard deviations for this distribution is 10,000, so about 10,000 Leapfrog steps would be needed in standard HMC with an identity covariance to obtain independent sample with each HMC update. Even in fairly regular landscapes such as the logistic regression landscapes from Section 9.6.2, the accuracy of the Leapfrog approximation degrades after more than a few hundred Leapfrog steps, and in practical problems, it is simple not possible to compensate for the differences in scale in the target distribution by using $L = 10,000$ Leapfrog updates with standard HMC. To sample effectively, it is necessary to take second order information into account.

Assume that the true position covariance Σ_2 is unknown and impossible to calculate directly, which is true in most practical situations. In this case, it is still possible to estimate Σ_2 from the previously sampled positions, and this approximate information can still facilitate fairly effective sampling. First, 40 positions were sampled using standard HMC with $\varepsilon = 0.000075$ and momentum covariance I_{100} . After obtaining some initial points, the simplified RMHMC algorithm was implemented using a LBFGS estimate of Σ_2 . The LBFGS recursion was started from an initial matrix $H_0 = \gamma I_{100}$ using the past 40 sampled positions.

Unfortunately, the raw LBFGS estimate cannot improve sampling significantly, since the true matrix Σ_2 is simply too large to be accurately estimated using only 40 points. However, after decomposition and adjustment, a useful relative can be obtained. When looking at the eigenvalues from the LBFGS estimate, one observes that the estimate for the smallest eigenvalue is very close to the true value of 0.0001^2 , and that the LBFGS estimate can identify several unconstrained directions with large eigenvalues. The estimates for the largest eigenvalue tended to be very inaccurate and to depend heavily on the value of γ selected. Between the largest and smallest eigenvalues, the majority of the remaining eigenvalues are unchanged by LBFGS and remain γ , which is unsurprising because a very small amount of data is being used to estimate a very large matrix.

Although the true covariance Σ_2 is unknown, in some situations it is reasonable to assume that only the first few largest eigenvalues of Σ_2 matter, and that most of the other eigenvalues are close to 0. This situation occurs when sampling a low-dimensional manifold in a high-dimensional space, which is common when sampling distributions defined over images or other complex data structures. The largest eigenvalues correspond to the relatively small number of unconstrained dimensions in the target distribution. Given some knowledge about the local structure of the eigenvalues, the raw

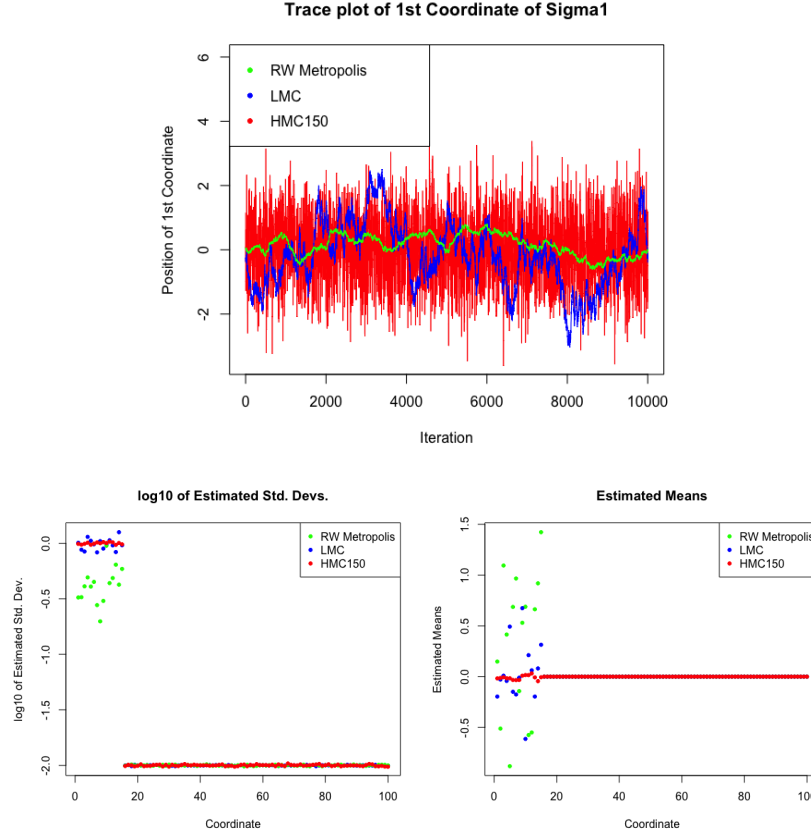


Figure 9.1: Simulation study using Σ_1 . The performance of the samplers is similar across the constrained dimensions, but RW Metropolis and LMC have difficulty effectively sampling the unconstrained dimensions, even though 150 iterations of each of these methods was counted as a single update in the figures. On the other hand, the capability of HMC to move across the sample space by reusing the same momentum for a large number of leapfrog iterations allows for very effective sampling.

LBFGS estimate can be adjusted to provide a more useful covariance.

Let H^* be the raw LBFGS estimated obtained from the past $J = 40$ HMC samples. Let $U\Lambda U^\top$ be the symmetric eigenvalue decomposition of H^* , where Λ is a diagonal matrix with eigenvalues $\lambda_1, \dots, \lambda_{100}$ sorted in decreasing order. Let $\lambda_i^* = \lambda_{100}$ for $i = K + 1, \dots, 100$ for some parameter K , which is the estimated number of unconstrained directions in the target distribution. The true value of K is 15, but in the experiment the conservative estimate $K = 10$ is used. The first K of the λ_i^* are equal to the original values. The momentum covariance is then given by $\Sigma^* = U\Lambda^*U^\top$. In theory, ε should be set to slightly less than 1 for any RMHMC method. However, due to the inaccuracy of the estimate of the largest standard deviation, which tends to be too large, ε should be set so that $\varepsilon\lambda_1 \approx 1$. The value of γ did not have too much effect on sampling, and values ranging from 0.000001 to 1 gave about the same results, as long as ε was set accordingly. Only $L = 1$ Leapfrog step was needed to achieve good results. The third method of Lemma 9.2 with $\sqrt{C}^{-1} = U(\Lambda^*)^{1/2}U^\top$ was used during implementation.

Two methods are presented for sampling Σ_2 : HMC with $L = 150$, and the simplified RMHMC algorithm described above. The simplified RMHMC algorithm using quasi-Newton information out-

performs standard HMC with only $L = 1$ Leapfrog step. The eigenvalue decomposition required at each iteration of the simplified RMHMC algorithm is computationally expensive, but it is necessary for good results, since the LBFGS estimate simply cannot estimate all of the true eigenvalues of Σ_2 with such a limited amount of data, so some adjustment of the eigenvalues is needed.

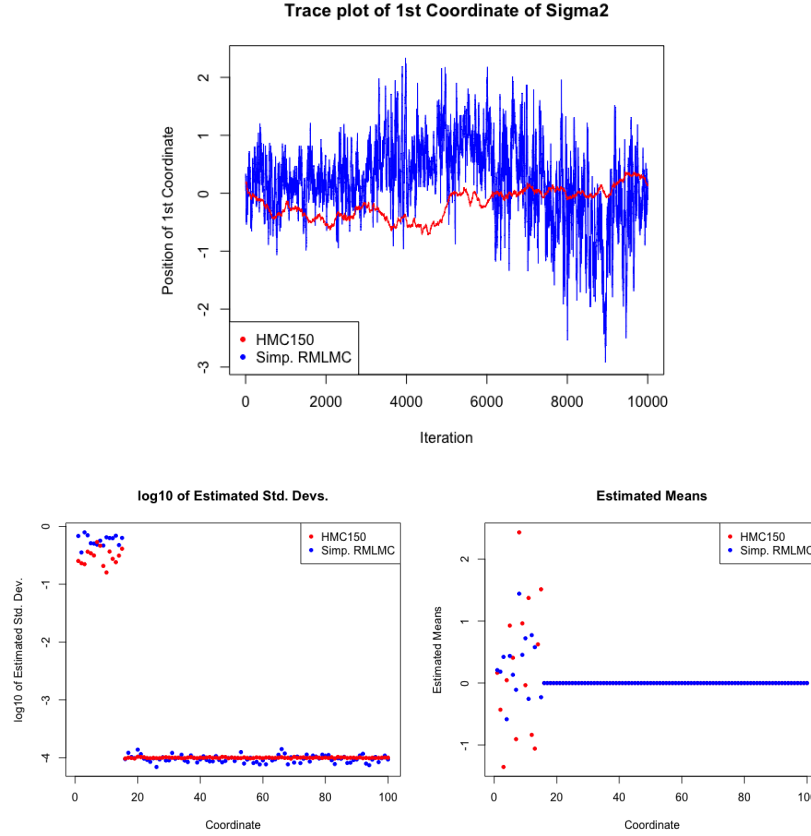


Figure 9.2: Simulation study using Σ_2 . HMC with $L = 150$ Leapfrog updates can no longer effectively sample in the unconstrained dimensions. However, using the simplified RMHMC algorithm with $L = 1$ and a covariance calculated as described above can sample much more effectively.

9.6.2 Sampling Logistic Regression Coefficients with RMHMC

Sampling coefficients from distributions defined by logistic regression is an ideal scenario for applying the full RMHC method, because the Fisher Information at any point in the state space can be given in closed form. In many practical situations (for example, L_1 -regularized regression) this is not the case. Given an $N \times P$ of observations X (each row gives a single case) and a binary 0 or 1 response Y and regularizing coefficient λ (treated as a given constant), the energy of P -length vector of coefficients β is given by

$$U(\beta) = -\log[L(X, Y|\beta, \lambda)p(\beta|\lambda)] = -\beta^\top X^\top Y + \sum_{j=1}^N \log(1 + e^{\beta^\top X_n^\top}) + \frac{\lambda}{2} \beta^\top \beta \quad (9.47)$$

where X_n is row n of the matrix X . The derivative of the energy function is

$$\frac{\partial U}{\partial \beta}(\beta) = -X^\top Y + X^\top S + \lambda \beta \quad (9.48)$$

where S is a length P vector with $S_n = \sigma(\beta^\top X_n^\top)$ where $\sigma(*)$ is the sigmoid function, and has Fisher Information

$$I(\beta) = E_{Y|X,\beta,\lambda} \left[\frac{\partial^2 U}{\partial \beta^2}(\beta) \right] = X^\top \Lambda X + \lambda I \quad (9.49)$$

where Λ is an $N \times N$ diagonal matrix with elements $\Lambda_{n,n} = \sigma(\beta^\top X_n^\top)(1 - \sigma(\beta^\top X_n^\top))$. Full RMHMC also requires the derivatives of $I(\beta)$, which are given by

$$\frac{\partial I(\beta)}{\partial \beta_i} = X^\top \Lambda V_i X \quad (9.50)$$

where V_i is diagonal matrix with elements $V_{i,(n,n)} = (1 - 2\sigma(\beta^\top X_n^\top))X_{n,i}$.

Presented below are the results of a study by Giorlami and Calderhead in [85] comparing the performance of RMHMC, traditional HMC, and other common methods when sampling regression coefficients. The authors used 6 different datasets with a binary response and features matrices of various sizes, and the results for 4 datasets are presented here. We give the results for 6 of the sampling algorithms studied by the authors: component-wise Metropolis-Hastings, LMC, HMC, RMHMC, RMLMC, and simplified RMLMC.

For LMC and HMC samplers, the momentum covariance $\Sigma = I_n$ was used. For all samplers, the step size ε was set so that acceptance rates were around 70%. The step size L for RMHMC and HMC was set so that $\varepsilon L \approx 3$, slightly larger than the largest marginal standard deviation, so that approximately independent samples should be obtained with each HMC iteration. The Hamiltonian dynamics defined by logistic regression are relatively well-behaved so L can be set to be quite large (several hundred) without a significant drop in acceptance rates. The number of leapfrog steps L is usually relatively small for RMHMC because nearly independent points can be obtained with each leapfrog update. For HMC, a large number of leapfrog updates are needed to compensate for the difference in scale between the smallest and largest marginal standard deviations. Simplified RMLMC corresponds to the simplified RMHMC algorithm where $\Sigma(\beta_t) = I(\beta_t)$, the Fisher information at the current point ($J = 1$) and $L = 1$ Leapfrog update. RMLMC is a slight variant of the RMHMC algorithm with $L = 1$. See the original paper for details.

All regression models included an intercept, so the number of coefficients P is one more than the number of columns of the data matrix. Each sampling run consisted of 5000 burn-in iterations and 5000 sampling iterations, and 10 trials were run for each sampling method.

Pima Indian Dataset, $N = 532$, $P = 8$

Method	Time (sec)	ESS(Min,Med,Max)	s/ESS(Min)	Rel. Speed
Metropolis	4.1	(14, 37, 201)	0.29	$\times 1.9$
LMC	1.63	(3, 10, 39)	0.54	$\times 1$
HMC	1499.1	(3149,3657,3941)	0.48	$\times 1.1$
RMLMC	4.4	(1124,1266,1409)	0.0039	$\times 138$
Simp. RMLMC	1.9	(1022,1185,1312)	0.0019	$\times 284$
RMHMC	50.9	(5000,5000,5000)	0.01	$\times 54$

Australian Credit Dataset, $N = 690$, $P = 14$

Method	Time (sec)	ESS(Min,Med,Max)	s/ESS(Min)	Rel. Speed
Metropolis	9.1	(15, 208, 691)	0.61	$\times 1$
LMC	No Conv.	-	-	-
HMC	No Conv.	-	-	-
RMLMC	11.8	(730, 872, 1033)	0.0162	$\times 37$
Simp. RMLMC	2.6	(459, 598, 726)	0.0057	$\times 107$
RMHMC	145.8	(4940, 5000, 5000)	0.023	$\times 26$

German Credit Dataset, $N = 1000$, $P = 24$

Method	Time (sec)	ESS(Min,Med,Max)	s/ESS(Min)	Rel. Speed
Metropolis	20.9	(10, 82, 601)	2.09	$\times 1$
LMC	2.7	(3, 5, 130)	0.9	$\times 2.6$
HMC	3161.6	(2707, 4201, 5000)	1.17	$\times 2$
RMLMC	36.2	(616, 769, 911)	0.059	$\times 39.6$
Simp. RMLMC	4.1	(463, 611, 740)	0.0009	$\times 260$
RMHMC	287.9	(4791, 5000, 5000)	0.06	$\times 39$

Caravan Dataset, $N = 5822$, $P = 86$

Method	Time (sec)	ESS(Min,Med,Max)	s/ESS(Min)	Rel. Speed
Metropolis	388.7	(3.8, 23.9, 804)	101.9	$\times 3.7$
LMC	17.4	(2.8, 5.3, 17.2)	6.2	$\times 59$
HMC	12,519	(33.8, 4032, 5000)	369.7	$\times 1$
RMLMC	305.3	(7.5, 21.1, 50.7)	305.3	$\times 1.2$
Simp. RMLMC	48.9	(7.5, 18.4, 44)	6.5	$\times 56$
RMHMC	45,760	(877, 1554, 2053)	52.1	$\times 7.1$

Ratio of Largest to Smallest Marginal Standard Deviations

Dataset	Pima	Australian	German	Caravan
Ratio	225	6404	303	236

A variety of useful instructive observations can be drawn from the results of these experiment. Following the original authors, the speed and relative speed of the algorithm are given based on the minimum ESS out of the 10 trials.

First, consider the performance of HMC and LMC across the 4 datasets. As mentioned earlier, the number of leapfrog steps L in HMC was tuned so that εL was greater than the largest observed standard deviation in the dataset, so HMC should in theory provide nearly independent samples, provided that the dynamics can be simulated accurately. Although slow, HMC does manage to achieve an ESS that is a significant proportion of the ideal ESS of 5000 in the Pima, German, and Caravan datasets. However, the authors found that neither HMC nor LMC converged to the stationary distribution in the Australian dataset.

This can be understood by referring to the table giving the ratios of largest to smallest marginal standard deviations. Recall that the ratio between the largest and smallest marginal standard deviations is the minimum number of Leapfrog steps that would be needed to reach an independent state in a single HMC update. In the Pima, German, and Caravan datasets, this ratio was about 200 to 300, meaning that between 200 to 300 leapfrog steps would be needed to reach an independent state using a trivial covariance matrix $\Sigma = I_n$. However, the Australian dataset has a ratio of over 6000 between the length of its largest and smallest constrained directions, so several thousand leapfrog steps are needed for each HMC update to reach an independent state. The Leapfrog discretization was not accurate enough to provide high acceptance rates for such large L , and the updates from the LMC sampler were too small to allow for effective sampling. The logistic regression

landscape is relatively well-behaved; in more complex landscapes, it might not be possible to use even 200 to 300 Leapfrog steps.

It is interesting to note that in general, LMC performs significantly worse than all of the other methods, except in the high-dimensional Caravan dataset, where it outperforms all other methods. LMC has been found to be an effective method for sampling very high-dimensional generative models of image patterns. LMC is an attractive choice in such situations because it scales well and it is quick to implement.

Now, consider the RMHMC methods. RMHMC and RMLMC have similar performance across the datasets, indicating that after local curvature is taken into account, the extra sampling power of RMHMC is evenly balanced by the faster speed of RMLMC. Simplified RMLMC outperforms the full RMHMC implementations across all datasets, providing evidence that simply taking into account local curvature information with no change in the HMC dynamics can be an effective way to implement and RMHMC approximation.

For the smaller datasets, the full RMHMC methods outperformed standard HMC methods, but the reverse was true for the Caravan dataset. Calculating the derivative of the Fisher Information in full RMHMC requires evaluating N^2 expressions along the diagonal in 9.49, which explains the poor scaling. In special cases, such as sampling from Gaussian processes, the Fisher Information can have a sparse structure, but in general it is unavailable or very expensive to compute, and full RMHMC is not a practical solution to sampling in complex high-dimensional landscapes.

9.6.3 The Alternating Backward Propagation (ABP) Algorithm

The ABP algorithm [90] is an extension of the well-known factor analysis model that can be used to generate realistic images from a simple low-dimensional latent distribution. The ABP algorithm, as the name suggests, has two phases. In the first phase, the latent factors of the set of training images are inferred using Langevin dynamics. In the second phase, the weights which transform the latent factors into images are updated based on the new latent factors. Since the latent factors are inferred during the training process, the ABP algorithm performs unsupervised learning. The ABP algorithm is closely related to the EM algorithm: the first phase corresponds to the E step of the EM algorithm, where expected values are evaluated based on the current parameters, and the second phase corresponds to the M step, where the parameters are adjusted to explain the expected factors.

Let Y be a D dimensional data vector (an image, for example) and let $Z \sim N(0, I_d)$ be the distribution of the latent factors. In the ABP algorithm, $Y \sim N(f(Z; W), \sigma^2 I_D)$ for a ConvNet function $f(*; W)$ with weights W . The weights W define the transformation from the latent space of Z to the image space of Y , and must be learned. In the traditional factor analysis model, $f(Z; W) = WZ + \varepsilon$, but using a ConvNet function allows for non-linearities in f through the inclusion of an activation function between layers. The ABP algorithm can be thought of as a recursive factor analysis model with the relation

$$Z_{l-1} = f_l(W_l Z_l + b_l) \quad (9.51)$$

where f_l is a non-linear activation (usually ReLu), (W_l, b_l) are the weights and bias from layer l of $W = \{W_l, b_l : l = 1, \dots, L\}$, $Z_L = Z$, and $Z_0 = f(Z; W)$. The layer of factors Z_{l-1} is a linear combination of the columns of W_l with coefficients Z_l , plus a shift and activation. If the ReLu activation is used, then $f(Z; W)$ is a piecewise linear function, where the boundaries between linear regions correspond to activation boundaries of f_l . This non-linear structure is essential for generating realistic images.

Since Z and $Y|Z$ are both multivariate normal, their joint energy function has the form

$$U(Y, Z; W) = \frac{1}{2\sigma^2} \|Y - f(Z; W)\|^2 + \frac{1}{2} \|Z\|^2$$

which is simply the sum of the Gaussian energy functions of Z and $Y|Z$. The energy function of the conditional variable $Z|Y$ is $U_{Z|Y=y; W}(z) = U(z, y; W)$, since the posterior distribution $Z|Y$ is

proportional to the joint distribution of Y and Z . As in the EM algorithm, given a set of complete observations $\{Y_i, Z_i\}_{i=1, \dots, N}$, it is straightforward to estimate the weights W through maximum likelihood by minimizing the complete data negative log likelihood

$$L(W, \{Z_i\}) = - \sum_{i=1}^N \log p(Y_i, Z_i; W) = \sum_{i=1}^N U(Y_i, Z_i)$$

by gradient descent. However, as in the EM algorithm, learning is unsupervised and the latent factors Z are unknown, so W must be learned by maximizing the observed data loglikelihood, which corresponds to minimizing function

$$L(W) = - \sum_{i=1}^N \log p(Y_i; W) = - \sum_{i=1}^N \log \int p(Y_i, Z_i; W) dZ_i$$

which integrates the latent factors out of the joint distribution. This loss cannot be computed directly, but the gradient of the loglikelihood can be rewritten as

$$\begin{aligned} \frac{\partial}{\partial W} \log p(Y; W) &= \frac{1}{p(Y; W)} \frac{\partial}{\partial W} p(Y; W) \\ &= \frac{1}{p(Y; W)} \frac{\partial}{\partial W} \int p(Y, Z; W) dZ \\ &= \int \left(\frac{1}{p(Y; W)} \frac{\partial}{\partial W} p(Y; W) \right) p(Z|Y; W) dZ \\ &= \int \left(\frac{\partial}{\partial W} \log p(Y; W) \right) p(Z|Y; W) dZ \\ &= -\mathbb{E}_{Z|Y; W} \left[\frac{\partial}{\partial W} U(Y, Z; W) \right] \end{aligned}$$

which shows that the gradient of the loss can be estimated by drawing MCMC samples of $Z|Y$, the latent factors conditioned on the observed data using the current weight W . LMC with a proposal covariance $\Sigma = I_d$ is used to sample from $Z|Y; W$, and the Langevin update equation is

$$Z^{t+1} = Z^t + \frac{\varepsilon^2}{2} \left(\frac{1}{\sigma^2} (Y - f(Z^t; W)) \frac{\partial}{\partial Z} f(Z^t; W) - Z^t \right) + \varepsilon U_t$$

for $U_t \sim \mathcal{N}(0, I_d)$ and step size ε , for $t = 1, \dots, T$ iterations. One Z_i is inferred for each observed image Y_i , and sampling is started from the Z_i of the previous inference phase at the beginning of the each inference phase. Once the Z_i have been sampled from $Z|Y; W$, the weights W can be updated

$$\frac{\partial}{\partial W} L(W) \approx \sum_{i=1}^N \frac{\partial}{\partial W} U(Y_i, Z_i; W) = \sum_{i=1}^N \frac{1}{\sigma^2} (Y_i - f(Z_i; W)) \frac{\partial}{\partial W} f(Z_i; W)$$

in the second phase of the algorithm. The inference phase uses a back-propagation gradient $\frac{\partial}{\partial Z} f(Z; W)$, while the learning phase uses a back-propagation gradient $\frac{\partial}{\partial W} f(Z; W)$. The calculations required to calculate $\frac{\partial}{\partial Z} f(Z; W)$ are also needed as part of the calculations required to obtain $\frac{\partial}{\partial W} f(Z; W)$, so both phases can be implemented in a similar way.

In practice, the Metropolis-Hastings step of the Langevin update can be ignored as long as ε is set low enough so that high acceptance rates are possible. A step size ε between 0.1 and 0.3 has been shown to work well in practice. Usually the number of updates T is set between 10 and 30. It

could be the case that more iterations are needed to reach a truly independent sample, especially if the ratio between the largest and smallest standard deviations of $Z|Y; W$ is large, as discussed previously. However, since a warm start is used, it is reasonable to believe that inferred Z_i should approximately follow $Z|Y; W$ if the previously inferred Z_i approximately follow $Z|Y; W'$ for a set of weights W' which are quite similar to the current weights W . Moreover, since the energy landscape itself is altered each time W is altered, there is also less reason to worry about becoming trapped in local modes during sampling, since the ABP algorithm is constantly adjusting the location of the modes during training.

Three different experiments are presented showing the capabilities of the ABP algorithm. See the original paper for details about tuning the models.

Experiment 1: Generating Texture Patterns. Let the input Z be a $\sqrt{d} \times \sqrt{d}$ dimensional images with each pixel following a standard normal distribution. The weights at each layer are given by convolutional filters with an upsampling factor of 2 at each layer. Once the filters are learned, it is straightforward to expand the network and generate larger texture patterns simply by increasing the size of Z and running the filter convolutions over the larger input. In the example below, Z was a 7×7 image recreating a 224×224 image during training, while during test Z was expanded to a 14×14 image generating a 448×448 image using exactly the same weights.



Figure 9.3: Experiment 1. Original images sized 224×224 next to synthesized images sized 448×448 .

Experiment 2: Generating Object Patterns. Generating object patterns is similar to generating texture patterns, except that the latent factor layer must be fully connected, and the input Z is thought of as a d -dimensional vector. The figures below show two different object patterns generated by the ABP algorithm: lion/tiger faces, and human faces. Interpolating between points in the latent space of the learned network gives non-linear interpolations in the image space which preserve structure far more than simple linear interpolation in the image space.

Experiment 3: Learning from Incomplete Data. In some situations, training images might be missing pixels due to corruption or occlusion. The ABP algorithm can learn a generator model for the complete image given a training set where certain pixels are labelled as missing. The only adjustment that needs to be made is to calculate the differences $\|Y_i - f(Z_i; W)\|^2$ by summing over the observed pixels of Y_i , and ignoring the unobserved pixels. The learned model can then accomplish three tasks: (1) Recover the missing pixels from the training images; (2) Recover the



Figure 9.4: Experiment 2. *Left:* A 9×9 discretization of the 2-dimensional latent space of a generative model for lion/tiger faces. The latent space has identifiable regions separating lions and tigers, and the interpolations between these regions smoothly transform a lion face to a tiger face. *Right:* Synthesized human faces from a generative model with 100 latent factors. The left images show 81 faces sampled from the learned model, and the right images show linear interpolations in the latent space between the faces on the four corners of the image.

missing pixels from the testing images; (3) Synthesize new images from the model.

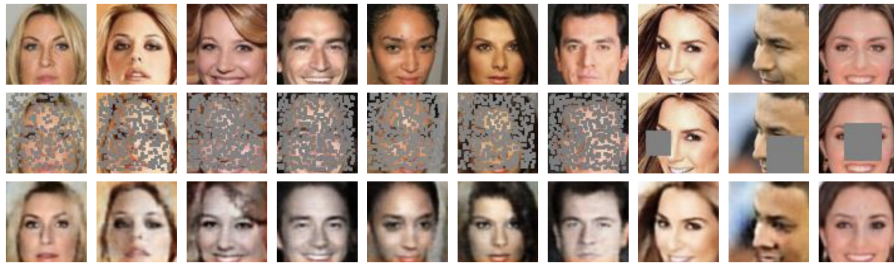


Figure 9.5: Experiment 3. *Top:* Original images. *Middle:* occluded training images. *Bottom:* Reconstructed training images.

Chapter 10

Stochastic Gradient for Learning

10.1 The Robbins-Monro Algorithm

The motivation for the Robbins-Monro algorithm comes from learning problems.

Example 10.1. Suppose we have a learning problem where we are given a large number of training examples $\mathbf{x}_i, i = \overline{1, N}$ and an objective function

$$L(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^N l(\mathbf{x}_i, \mathbf{w})$$

obtained as the average of a function $l(\cdot, \mathbf{w})$ over the training examples. The function $l(\mathbf{x}_i, \mathbf{w})$ is assumed differentiable in \mathbf{w} for all i .

One could minimize this objective function by gradient descent, however each gradient computation $\nabla L(\mathbf{w})$ is the average of the gradients at the observations $\nabla l(\mathbf{x}_i, \mathbf{w})$, which can be computationally expensive when the number of training examples is large. The per-observation gradients $\nabla_{\mathbf{w}} l(\mathbf{x}_i, \mathbf{w})$ can be considered noisy versions of the true gradient $\nabla L(\mathbf{w})$ with $E[\nabla_{\mathbf{w}} l(\mathbf{x}, \mathbf{w})] = \nabla L(\mathbf{w})$. Denoting $F(\mathbf{w}) = \nabla L(\mathbf{w})$, the minimization problem can be reduced to finding a root of $F(\mathbf{w})$ when we are given noisy measurements $G(\mathbf{w}) = \nabla_{\mathbf{w}} l(\mathbf{x}_i, \mathbf{w})$ of $F(\mathbf{w})$ with $E[G(\mathbf{w})] = F(\mathbf{w})$.

The Robbins-Monro Algorithm [168] is a generic method for finding a root of a function $F(\mathbf{w}) : \mathbb{R}^d \rightarrow \mathbb{R}^d$ where the function cannot be measured exactly, but noisy observations $G(\mathbf{w})$ can be obtained that have as expected value the function $F(\mathbf{w})$. We assume that $F(\mathbf{w}) = 0$ has a unique root at $\mathbf{w} = \theta \in \mathbb{R}^d$ and the noisy observations are in the form of a random variable $G(\mathbf{w})$ with $E[G(\mathbf{w})] = F(\mathbf{w})$.

The Robbins-Monro algorithm finds a sequence of approximations:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n G(\mathbf{w}_n) \tag{10.1}$$

where $\gamma_1, \gamma_2, \dots$ is a sequence of positive learning rates (step sizes).

The convergence analysis of the algorithm is limited to the 1-dimensional case $d = 1$.

Theorem 10.1 (Robbins-Monro). *The sequence w_n converges in L^2 (hence in probability) to θ if the following conditions are met:*

1. $G(w)$ is uniformly bounded in the sense that there exists $C < \infty$ such that $P(|G(w)| \leq C) = 1$.
2. $F(w)$ is non-decreasing, differentiable and $F'(\theta) > 0$.

3. The sequence γ_n satisfies $\sum_{n=0}^{\infty} \gamma_n = \infty$ and $\sum_{\gamma=0}^{\infty} \gamma_n^2 < \infty$.

Multi-dimensional versions are limited by many restrictive assumptions. A recent explicit version [144] considers the case when the noisy observations $G(\mathbf{w}_n)$ are gradients of convex differentiable functions $f_n : \mathbb{R}^d \rightarrow \mathbb{R}$, thus $G(\mathbf{w}_n) = \nabla f_n(\mathbf{w}_n)$.

Theorem 10.2 (Moulines, 2011). *Assume the following conditions are met*

H1) *There exists a differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ such that*

$$E[\nabla f_n(\mathbf{w})] = \nabla f(\mathbf{w}), \quad \forall n \geq 1 \quad \text{with probability 1}$$

H2) *For all $n \geq 1$ f_n is almost surely convex, differentiable and*

$$\forall \mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d, E(\|\nabla f_n(\mathbf{w}_1) - \nabla f_n(\mathbf{w}_2)\|^2) \leq L^2 \|\mathbf{w}_1 - \mathbf{w}_2\|^2 \quad \text{with probability 1}$$

H3) *The function f is strongly convex with respect to the norm $\|\cdot\|$ with some constant $\mu > 0$:*

$$\forall \mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d, f(\mathbf{w}_1) \geq f(\mathbf{w}_2) + (\mathbf{w}_1 - \mathbf{w}_2) \nabla f(\mathbf{w}_2) + \frac{\mu}{2} \|\mathbf{w}_1 - \mathbf{w}_2\|^2$$

H4) *There exists $\sigma > 0$ such that $\forall n \geq 1, E(\|\nabla f_n(\theta)\|^2) \leq \sigma$.*

Let $\delta_0 = \|\mathbf{w}_0 - \theta\|^2$, $\varphi_\beta(t) = \frac{t^\beta - 1}{\beta}$ and $\gamma_n = Cn^{-\alpha}$ for some $\alpha \in [0, 1]$. Then the sequence from eq. (10.1) satisfies

$$E\|\mathbf{w}_n - \theta\|^2 \leq 2 \exp \left[4L^2 C^2 \varphi_{1-2\alpha}(n) - \frac{\mu C}{2} n^{1-\alpha} \right] \left(\delta_0 + \frac{\sigma^2}{L^2} \right) + \frac{4C\sigma^2}{\mu n^\alpha}$$

when $\alpha < 1$. If $\alpha = 1$ then:

$$E\|\mathbf{w}_n - \theta\|^2 \leq \frac{\exp(2L^2 C^2)}{n^{\mu C}} \left(\delta_0 + \frac{\sigma^2}{L^2} \right) + 2 \frac{C^2 \sigma^2}{n^{\mu C/2}} \varphi_{\mu C/2-1}(n).$$

For the learning example 10.1 the Robbins-Monro algorithm is called stochastic gradient descent and has different versions:

1. The online version $\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \nabla_{\mathbf{w}} l(\mathbf{x}_i, \mathbf{w}_n)$ that uses a single example for each update.
2. The mini-batch version

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma_n \frac{1}{B} \sum_{i \in B_n} \nabla_{\mathbf{w}} l(\mathbf{x}_i, \mathbf{w}_n)$$

that uses a set of examples (mini-batch) B_n of size $|B_n| = B$ for each update.

Convergence time is proportional to the condition number $\kappa = \lambda_{\max}/\lambda_{\min}$ of the Hessian matrix $H_{ij} = \frac{\partial^2 L(\mathbf{w})}{\partial w_i \partial w_j}$ at the minimum [113].

The full gradient iteration (with batch size equal to n) has *linear* convergence in the sense that $L(\mathbf{w}^k) - L(\theta) = O(\rho^k)$ where $\rho < 1$ depends on the condition number κ (from [149], Theorem 2.1.15). The online version has a sub-linear convergence $E[L(\mathbf{w}^k)] - L(\theta) = O(1/k)$ [171] but each iteration is n times faster.

Example 10.2. Another example is for learning Markov Random Field (MRF) models

$$p(\mathbf{x}|\mathbf{w}) = \frac{1}{Z(\mathbf{w})} \exp(-\langle \mathbf{w}, U(\mathbf{x}) \rangle)$$

where $\mathbf{w} \in \mathbb{R}^d$ are the model parameters to be learned, $\mathbf{x} \in \Omega \subset \mathbf{R}^M$ is the space over which the MRF is defined, $U(\mathbf{x}) : \Omega \rightarrow \mathbb{R}^d$ are sufficient statistics and $Z(\mathbf{w})$ is the partition function

$$Z(\mathbf{w}) = \int_{\Omega} \exp(-\langle \mathbf{w}, U(\mathbf{x}) \rangle) d\mathbf{x}$$

It is well known that the log likelihood function $L(\mathbf{x}; \mathbf{w}) = \ln p(\mathbf{x}|\mathbf{w})$ is concave in \mathbf{w} , so the maximum likelihood solution over a set of instances $\mathbf{x}_i, i = \overline{1, n}$ satisfies:

$$\frac{\partial}{\partial \mathbf{w}} \sum_{i=1}^N L(\mathbf{x}_i; \mathbf{w}) = 0$$

which gives

$$E_{\mathbf{w}}(U) - \bar{U} = 0, \text{ with } \bar{U} = \frac{1}{N} \sum_{i=1}^N U(\mathbf{x}_i)$$

where $E_{\mathbf{w}}(U)$ is the expectation taken with respect to $p(\mathbf{x}|\mathbf{w})$. In this case the expectation $E_{\mathbf{w}}(U)$ cannot be computed exactly but noisy approximations can be obtained by Monte Carlo simulations.

The Robbins-Monro algorithm in this case first obtains a number k_n of Monte Carlo samples $\Omega \ni \mathbf{x}_j^n \sim p(\mathbf{x}|\mathbf{w}_n), j = \overline{1, k_n}$ from the current model, and updates the parameters as follows:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \frac{\gamma_n}{k_n} \sum_{j=1}^{k_n} [U(\mathbf{x}_j^n, \mathbf{w}_n) - \bar{U}]$$

10.2 Parameter estimation methods for Gibbs/MRF models

Let \mathbf{I}_{Λ} be an image defined on a lattice Λ , and $\mathbf{I}_{\partial\Lambda}$ its boundary conditions. $\partial\Lambda$ is the neighborhood of Λ . Let $\mathbf{h}(\mathbf{I}_{\Lambda}|\mathbf{I}_{\partial\Lambda})$ be the feature statistics of \mathbf{I}_{Λ} under boundary conditions $\mathbf{I}_{\partial\Lambda}$. For example, $\mathbf{h}()$ are histograms of filtered images [219]. Without loss of generality, a Gibbs model is of the following form (see [222]),

$$p(\mathbf{I}_{\Lambda}|\mathbf{I}_{\partial\Lambda}; \boldsymbol{\beta}) = \frac{1}{Z(\mathbf{I}_{\partial\Lambda}, \boldsymbol{\beta})} \exp\{-\langle \boldsymbol{\beta}, \mathbf{h}(\mathbf{I}_{\Lambda}|\mathbf{I}_{\partial\Lambda}) \rangle\}. \quad (10.2)$$

In equation (10.2), $\boldsymbol{\beta}$ is a vector valued parameter corresponding to the Julesz ensemble on infinite images [208], and it is invariant to the sizes and shapes of Λ . So $\boldsymbol{\beta}$ can be estimated on an arbitrary Λ .

10.2.1 Learning in Gibbsian fields – a common framework

In learning Gibbs models, we are given an observed image $\mathbf{I}_{\Lambda}^{\text{obs}}$, where Λ may have many disconnected components to account for multiple observations. Equally we may break Λ into smaller patches $\mathbf{I}_{\Lambda_i}^{\text{obs}}, i = 1, 2, \dots, M$ on lattices Λ_i of arbitrary shapes and sizes. These patches may overlap with each other. Then $\boldsymbol{\beta}$ is learned by maximizing a log-likelihood,

$$\boldsymbol{\beta}^* = \arg \max_{\boldsymbol{\beta}} \mathcal{G}(\boldsymbol{\beta}), \quad \text{with } \mathcal{G}(\boldsymbol{\beta}) = \sum_{i=1}^M \log p(\mathbf{I}_{\Lambda_i}^{\text{obs}}|\mathbf{I}_{\partial\Lambda_i}^{\text{obs}}; \boldsymbol{\beta}). \quad (10.3)$$

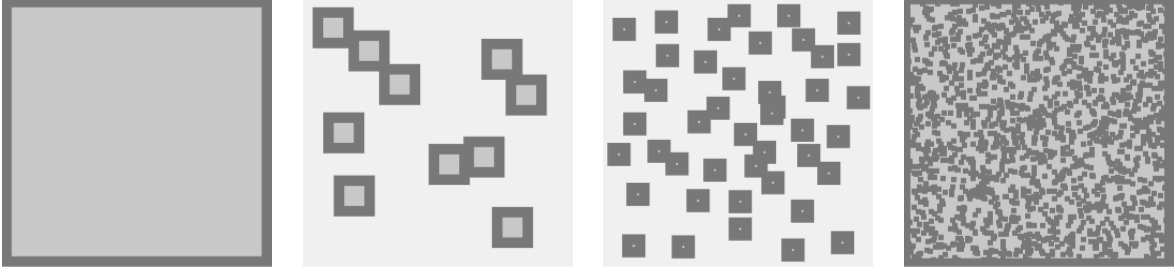


Figure 10.1: Various choices of Λ_i , $i = 1, 2, \dots, M$. The bright pixels are in foreground Λ_i which are surrounded by dark background pixels in $\partial\Lambda_i$. a). likelihood, b). patch likelihood (or satellite) likelihood, c). pseudo likelihood, d). partial likelihood.

We show that existing Gibbs learning algorithms are unified as ML-estimators and they differ in the following two choices.

Choice I: The number, sizes, and shapes of the *foreground* patches Λ_i , $i = 1, \dots, M$.

Figure 10.1 displays four typical choices for Λ_i . The bright pixels are in the *foreground* Λ_i , $i = 1, 2, \dots, M$, which are surrounded by dark pixels in the *background* $\partial\Lambda_i$, $i = 1, 2, \dots, M$. In the first three cases, Λ_i are square patches with $m \times m$ pixels. In one extreme, Figure 10.1.a chooses one largest patch denoted by Λ_1 , i.e. $M = 1$ and $m = N - 2w$ with w being the width of the boundary. $\mathcal{G}(\beta)$ is called the *log-likelihood*, and it is adopted by the stochastic gradient [213, 222] and MCMCMLE [53, 80, 81] methods. In the other extreme, Figure 10.1.c chooses the minimum patch size $m = 1$ and $\mathcal{G}(\beta)$ is called the *log-pseudo-likelihood*, used in the maximum pseudo-likelihood estimation (MPLE) [15]. Figure 10.1.b is an example between the two extremes and $\mathcal{G}(\beta)$ is called the *log-patch-likelihood*. In the fourth case, Figure 10.1.d chooses only one ($M = 1$) irregular-shaped patch, denoted by Λ_1 , where Λ_1 is a set of randomly selected pixels with the rest pixels being the background $\partial\Lambda_1$, and $\mathcal{G}(\beta)$ is called the *log-partial-likelihood*. In Figures 10.1.b and c, a foreground pixel may serve as background in different patches. It is straightforward to prove that maximizing $\mathcal{G}(\beta)$ leads to a consistent estimator for all four choices [83].

The flexibility of likelihood function distinguishes Gibbs learning from the problem of estimating partition functions [97, 163, 164]. The latter computes the “pressure” on a large lattice in order to overcome boundary effects.

Choice II: The reference models used for estimating the partition functions.

For a chosen foreground and log-likelihood function, the second step is to approximate the partition functions $Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}}, \beta)$ for each Λ_i , $i = 1, \dots, M$ by Monte Carlo integration using a reference model at β_o .

$$\begin{aligned}
Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}}, \beta) &= \int \exp\{-\langle \beta, \mathbf{h}(\mathbf{I}_{\Lambda_i} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \rangle\} d\mathbf{I}_{\Lambda_i}, \\
&= Z(\mathbf{I}_{\partial\Lambda_i}, \beta_o) \int \exp\{-\langle \beta - \beta_o, \mathbf{h}(\mathbf{I}_{\Lambda_i} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \rangle\} dp(\mathbf{I} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}; \beta_o) \\
&\approx \frac{Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}}, \beta_o)}{L} \sum_{j=1}^L \exp\{-\langle \beta - \beta_o, \mathbf{h}(\mathbf{I}_{ij}^{\text{syn}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \rangle\}.
\end{aligned} \tag{10.4}$$

$\mathbf{I}_{ij}^{\text{syn}}$, $j = 1, 2, \dots, L$ are *typical* samples from the reference model $p(\mathbf{I}_{\Lambda_i} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}; \beta_o)$ for each patch $i = 1, 2, \dots, M$.

Since $\frac{Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}}, \beta_o)}{L}$, $i = 1, 2, \dots, M$ are independent of β , we can maximize the estimated log-

likelihood $\mathcal{G}(\beta)$ by gradient descent. This leads to

$$\frac{d\beta}{dt} = \sum_{i=1}^M \left\{ \sum_{j=1}^L \omega_{ij} \mathbf{h}(\mathbf{I}_{ij}^{\text{syn}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) - \mathbf{h}(\mathbf{I}_{\Lambda_i}^{\text{obs}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \right\}. \quad (10.5)$$

ω_{ij} is the weight for sample $\mathbf{I}_{ij}^{\text{syn}}$,

$$\omega_{ij} = \frac{\exp\{-\langle \beta - \beta_o, \mathbf{h}(\mathbf{I}_{ij}^{\text{syn}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \rangle\}}{\sum_{j'=1}^L \exp\{-\langle \beta - \beta_o, \mathbf{h}(\mathbf{I}_{ij'}^{\text{syn}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \rangle\}}.$$

The selection of the reference models $p(\mathbf{I}_{\Lambda_i} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}; \beta_o)$ depends on the sizes of the patches Λ_i , $i = 1, \dots, M$. In general importance sampling is only valid when the two distributions $p(\mathbf{I}_{\Lambda_i} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}; \beta_o)$ and $p(\mathbf{I}_{\Lambda_i} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}; \beta)$ heavily overlap. In one extreme case $m = 1$, the MPLE method [15] selects $\beta_o = 0$ and $p(\mathbf{I}_{\Lambda_i} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}; \beta_o)$ a uniform distribution. In this case $Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}}; \beta)$ can be computed exactly. In the other extreme case for a large foreground $m = N - 2w$, the stochastic gradient and the MCMCMLE methods have to choose $\beta_o = \beta$ in order to obtain sensible approximations. Thus, both methods must sample $p(\mathbf{I}; \beta)$ iteratively starting from $\beta_o = 0$. This is the algorithm adopted in learning the FRAME models [222].

To summarize, Figure 10.2 illustrates two factors that determine the accuracy and speed of learning β . These curves are verified through experiments in section 10.2.3 (see Figure 10.7). The horizontal axis is the size of an individual foreground lattice $|\Lambda_i|$.

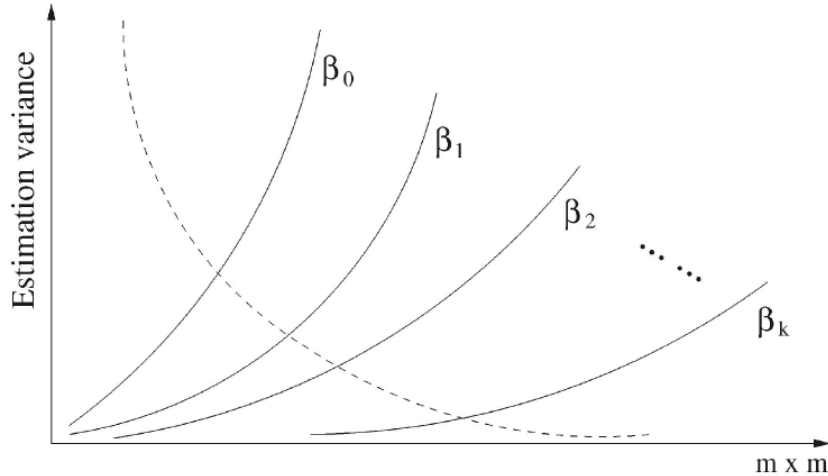


Figure 10.2: Estimation variances for various selections of patch sizes $m \times m$ and reference models β_o . The dashed curve shows the inverse Fisher's information which decreases as $m \times m$ increases. The solid curves show the variances in the importance sampling for a sequence of models approaching β .

1. *The variances of MLE or inverse Fisher information.* Let $\hat{\beta}(\mathbf{I}^{\text{obs}})$ be the estimator maximizing $\mathcal{G}(\beta)$ and let β^* be the optimal solution. The dashed curve in Figure 10.2 illustrates the variance,

$$E_f[(\hat{\beta}(\mathbf{I}^{\text{obs}}) - \beta^*)^2],$$

where $f(\mathbf{I})$ is a underlying distribution representing the Julesz ensemble. For choices shown in Figure 10.1, if we fix the total number of foreground pixels $\sum_{i=1}^M |\Lambda_i|$, then the variance (or estimation error) decreases as the patch size (diameter of the hole) increases.

2. *The variance of estimating Z by Monte Carlo integration $E_p[(\hat{Z} - Z)^2]$.* For a given reference model $\beta_o = \beta_i, i = 1, 2, \dots, k$ (see solid curves in Figure 10.2), this estimation error increases with the lattice sizes. Therefore, for very large patches, such as $m = 200$, we must construct a sequence of reference models to approach β , $\beta_0 = 0 \rightarrow \beta_1 \rightarrow \beta_2 \rightarrow \dots \rightarrow \beta_k \rightarrow \beta$. This is the major reason for why the stochastic gradient algorithm was so slow in FRAME [222]!

10.2.2 Three new algorithms

The analysis in previous section leads to three new algorithms by selecting likelihoods that trade-off between the two factors, and the third algorithm improve accuracy by pre-computed reference models.

Algorithm I: Maximizing partial likelihood.

We choose a lattice shown in Figure 10.1.d by choosing at random a certain percentage (say 30%) of pixels as foreground Λ_1 and the rest are treated as background Λ/Λ_1 . We define a *log-partial-likelihood*

$$\mathcal{G}_1(\beta) = \log p(\mathbf{I}_{\Lambda_1}^{\text{obs}} | \mathbf{I}_{\Lambda/\Lambda_1}^{\text{obs}}; \beta).$$

Maximizing $\mathcal{G}_1(\beta)$ by gradient descent, we update β iteratively.

$$\frac{d\beta}{dt} = E_{p(\mathbf{I}_{\Lambda_1} | \mathbf{I}_{\Lambda/\Lambda_1}^{\text{obs}}; \beta)}[\mathbf{h}(\mathbf{I}_{\Lambda_1} | \mathbf{I}_{\Lambda/\Lambda_1}^{\text{obs}})] - \mathbf{h}(\mathbf{I}_{\Lambda_1}^{\text{obs}} | \mathbf{I}_{\Lambda/\Lambda_1}^{\text{obs}}). \quad (10.6)$$

This algorithm follows the same procedure as the original method in FRAME [222]. It trades off between accuracy and speed in a better way than the original algorithm in FRAME [222]. The log-partial-likelihood has lower Fisher information than the log-likelihood, however our experiments demonstrate that it is about 25 times faster than the original minimax learning method without losing much accuracy. We observed that the reason for this speedup is that the original sampling method [222] spends a major portion of its time synthesizing $\mathbf{I}_{\Lambda_1}^{\text{syn}}$ under “non-typical” boundary conditions starting with white noise images. In contrast, the new algorithm works on typical boundary condition $\mathbf{I}_{\Lambda/\Lambda_1}^{\text{obs}}$ where the probability mass of the Gibbs model $p(\mathbf{I}; \beta)$ is focused on. The speed appears to be decided by the diameter of the foreground lattice measured by the maximum circle that can fit in the foreground lattice.

Algorithm II: Maximizing patch likelihood.

Algorithm II chooses a set of M overlapping patches from $\mathbf{I}_{\Lambda}^{\text{obs}}$ and “dig” a hole Λ_i on each patch as Figure 10.1.b shows. Thus we define a *patch log-likelihood*

$$\mathcal{G}_2(\beta) = \sum_{i=1}^M \log p(\mathbf{I}_{\Lambda_i}^{\text{obs}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}; \beta).$$

Maximizing $\mathcal{G}_2(\beta)$ by gradient descent, we update β iteratively as Algorithm I does.

$$\frac{d\beta}{dt} = \sum_{i=1}^M \mathbf{h}(\mathbf{I}_{\Lambda_i}^{\text{syn}} | \mathbf{I}_{\Lambda/\Lambda_i}^{\text{obs}}) - \sum_{i=1}^M \mathbf{h}(\mathbf{I}_{\Lambda_i}^{\text{obs}} | \mathbf{I}_{\Lambda/\Lambda_i}^{\text{obs}}). \quad (10.7)$$

In comparison with algorithm I, the diameters of the lattices are evenly controlled. Algorithm I has similar performance as algorithm I.

Algorithm III: Maximizing satellite likelihood

Both algorithms I and II still need to synthesize images on-line, which is a computationally intensive task. Now we propose an third algorithm which may approximately compute β in the speed of a few seconds without synthesizing images on-line.

We select a set of reference models in the exponential family Ω to which the Gibbs model $p(\mathbf{I}; \beta)$ belongs,

$$\mathcal{R} = \{p(\mathbf{I}; \beta_j) : \beta_j \in \Omega, j = 1, 2, \dots, s.\}$$

We sample (or synthesize) one large typical image $\mathbf{I}_j^{\text{syn}} \sim p(\mathbf{I}; \beta_j)$ for each reference model off-line. These reference models estimate β in Ω from different “viewing angles”. By analogy to the global positioning system, we call the reference models the “satellites”.

The *log-satellite-likelihood* is defined as

$$\mathcal{G}_3(\beta) = \mathcal{G}_3^{(1)}(\beta; \beta_1) + \mathcal{G}_3^{(2)}(\beta; \beta_2) + \cdots + \mathcal{G}_3^{(s)}(\beta; \beta_s), \quad (10.8)$$

where each satellite contributes one log-likelihood approximation,

$$\mathcal{G}_3^{(j)}(\beta; \beta_j) = \sum_{i=1}^M \log \frac{1}{\hat{Z}_i^{(j)}} \exp\{-\langle \beta, \mathbf{h}(\mathbf{I}_{\Lambda_i}^{\text{obs}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \rangle\}. \quad (10.9)$$

Following the importance sampling method in equation (10.4), we estimate $Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}}, \beta)$ by Monte Carlo integration.

$$\hat{Z}_i^{(j)} = \frac{Z(\mathbf{I}_{\partial\Lambda_i}^{\text{obs}}, \beta_j)}{L} \sum_{\ell=1}^L \exp\{-\langle \beta - \beta_j, \mathbf{h}(\mathbf{I}_{ij\ell}^{\text{syn}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \rangle\}. \quad (10.10)$$

Notice that for every hole Λ_i and for every reference model $p(\mathbf{I}; \beta_j)$, we have a set of L synthesized patches $\mathbf{I}_{ij\ell}^{\text{syn}}$ to fill the hole:

$$H_{ij}^{\text{syn}} = \{\mathbf{I}_{ij\ell}^{\text{syn}}; \ell = 1, 2, \dots, L, \forall i, j\}.$$

There are two ways for generating H_{ij}^{syn} .

1. Sampling $\mathbf{I}_{ij\ell}^{\text{syn}} \sim p(\mathbf{I}_{\Lambda_i} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}; \beta_j)$ – using the conditional distribution. This is expensive and has to be computed on-line.
2. Sampling $\mathbf{I}_{ij\ell}^{\text{syn}} \sim p(\mathbf{I}_{\Lambda_i}; \beta_j)$ – using the marginal distribution. In practice, this is just to fill the holes with randomly selected patches from the synthesized image $\mathbf{I}_j^{\text{syn}}$ computed off-line.

In our experiments, we tried both cases and we found that differences are very little for middle sizes $m \times m$ lattices, say $4 \leq m \leq 13$.

Maximizing $\mathcal{G}_3(\beta)$ by gradient ascent, we have,

$$\frac{d\beta}{dt} = \sum_{j=1}^s \left\{ \sum_{i=1}^M \left[\sum_{\ell=1}^L \omega_{ij} \mathbf{h}(\mathbf{I}_{ij\ell}^{\text{syn}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) - \mathbf{h}(\mathbf{I}_{\Lambda_i}^{\text{obs}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \right] \right\} \quad (10.11)$$

ω_{ij} is the weight for sample $\mathbf{I}_{ij\ell}^{\text{syn}}$,

$$\omega_{ij\ell} = \frac{\exp\{-\langle \beta - \beta_j, \mathbf{h}(\mathbf{I}_{ij\ell}^{\text{syn}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \rangle\}}{\sum_{\ell'=1}^L \exp\{-\langle \beta - \beta_j, \mathbf{h}(\mathbf{I}_{ij\ell'}^{\text{syn}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}}) \rangle\}}$$

Equation (10.11) converges in the speed of seconds for an average texture model.

However, we should be aware of the risk that the log-satellite-likelihood $\mathcal{G}_3(\beta)$ may not be upper bounded. It is almost surely not upper bounded for the MCMCMLE method. This case occurs when $\mathbf{h}(\mathbf{I}_{\Lambda_i}^{\text{obs}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}})$ cannot be described by a linear combination of the statistics of the sampled patches $\sum_{\ell=1}^L \omega_{ij} \mathbf{h}(\mathbf{I}_{ij\ell}^{\text{syn}} | \mathbf{I}_{\partial\Lambda_i}^{\text{obs}})$. When this occurs, β does not converge. We handle this problem by including the observed patch $\mathbf{I}_{\Lambda_i}^{\text{obs}}$ in H_{ij}^{syn} , therefore the satellite likelihood is always upper bounded. Intuitively, let $\mathbf{I}_{ij1}^{\text{syn}} = \mathbf{I}_{\Lambda_i}^{\text{obs}}$, β is learned so that the conditional probabilities $\omega_{ij1} \rightarrow 1$ and $\omega_{ij\ell} \rightarrow 0$, $\forall \ell \neq 1$. Since ℓ is often very large, say $\ell = 20$, adding one extra sample will not screw the sample set.

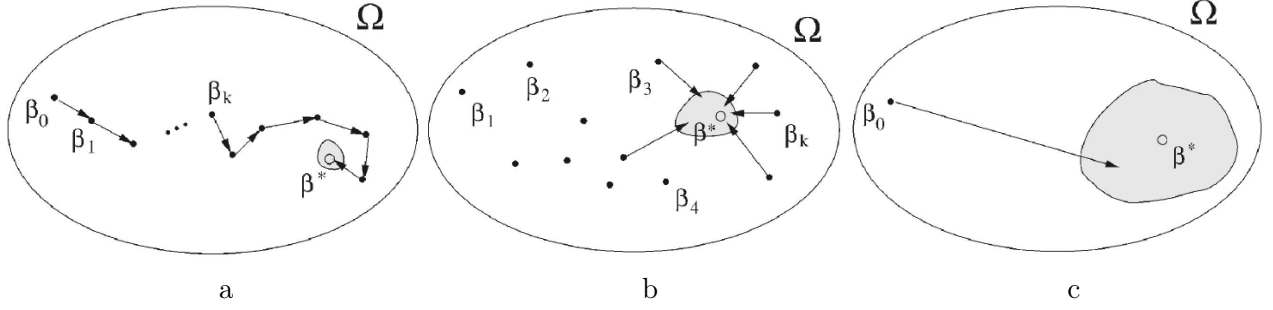


Figure 10.3: The shadow areas around β^* illustrates the variance of the estimated β or efficiency of the log-likelihood functions. a). Stochastic gradient, and algorithm I and II generate a sequence of satellites on-line to approach β^* closely, m can be small or large. b). The maximum satellite likelihood estimator uses a general set of satellites computed off-line, and can be updated incrementally. This can be used for small size m . c). MPLE uses a single satellite: $\beta_o = 0$.

To summarize, we compare existing algorithms and the newly proposed algorithms from the perspective of estimating β^* in Ω , and divide them into three groups. Figure 10.3 illustrates the comparison where the ellipse stands for the space Ω and each Gibbs model is represented by a single point.

Group 1: As Figure 10.3.a illustrates, the maximum likelihood estimators (including stochastic gradient and MCMCMLE) and the maximum partial/patch likelihood estimators generate and sample a sequence of “satellites” $\beta_0, \beta_1, \dots, \beta_k$ on-line. These satellites get closer and closer to β^* (supposed truth). The shadow area around β^* represents the uncertainty in computing β , whose size can be measured by the Fisher information.

Group 2: As Figure 10.3.c shows, the maximum pseudo likelihood estimator uses a uniform model $\beta_o = 0$ as a “satellite” to estimate any model, and thus has large variance.

Group 3: The maximum satellite likelihood estimators in Figure 10.3.b use a general set of satellites which are pre-computed and sampled off-line. To save time, one may select a small subset of satellites for computing a given model. One can choose satellites based on the differences $\mathbf{h}(\mathbf{I}_j^{\text{syn}})$ and $\mathbf{h}(\mathbf{I}^{\text{obs}})$. The smaller the differences are, the closer the satellite is to the estimated model, and thus better approximation. Another criterion is that these satellite should be distributed evenly around β^* to obtain good estimation.

10.2.3 Experiments

In this section, we evaluate the performance of various algorithms in the context of learning Gibbs models for textures. We use 12 filters including an intensity filter, two gradient filters, three Laplacian of Gaussian filters and six Gabor filters at a fixed scale and different orientations. Thus $\mathbf{h}(\mathbf{I})$ includes 12 histograms of filter responses and each histogram has 12 bins. So β has 12×11 free parameters.

We choose 15 natural texture images. For each texture, we use the stochastic gradient algorithm [222] to learn β which is treated as ground truth β^* for comparison. In this way, we also obtained 15 satellites with 15 synthesized images $\mathbf{I}_j^{\text{syn}}$ computed off-line.

Experiment I: Comparison of five algorithms.

In the first experiment, we compare the performance of five algorithms in texture synthesis. Figure 10.4 demonstrate 3 texture patterns of 128×128 pixels. For each row, the first column is the synthesized image (ground truth) using a stochastic gradient method used in the FRAME model [222], the other four images are respectively synthesized images using maximum pseudo-likelihood, maximum satellite likelihood, maximum patch likelihood and maximum partial likelihood. For the last three algorithms we fixed the total number of foreground pixels to 5,000. The patch size is fixed to 5×5 pixels for patch likelihoods and satellite likelihoods. We select 5 satellites out of the

rest 14 pre-computed models for each texture.

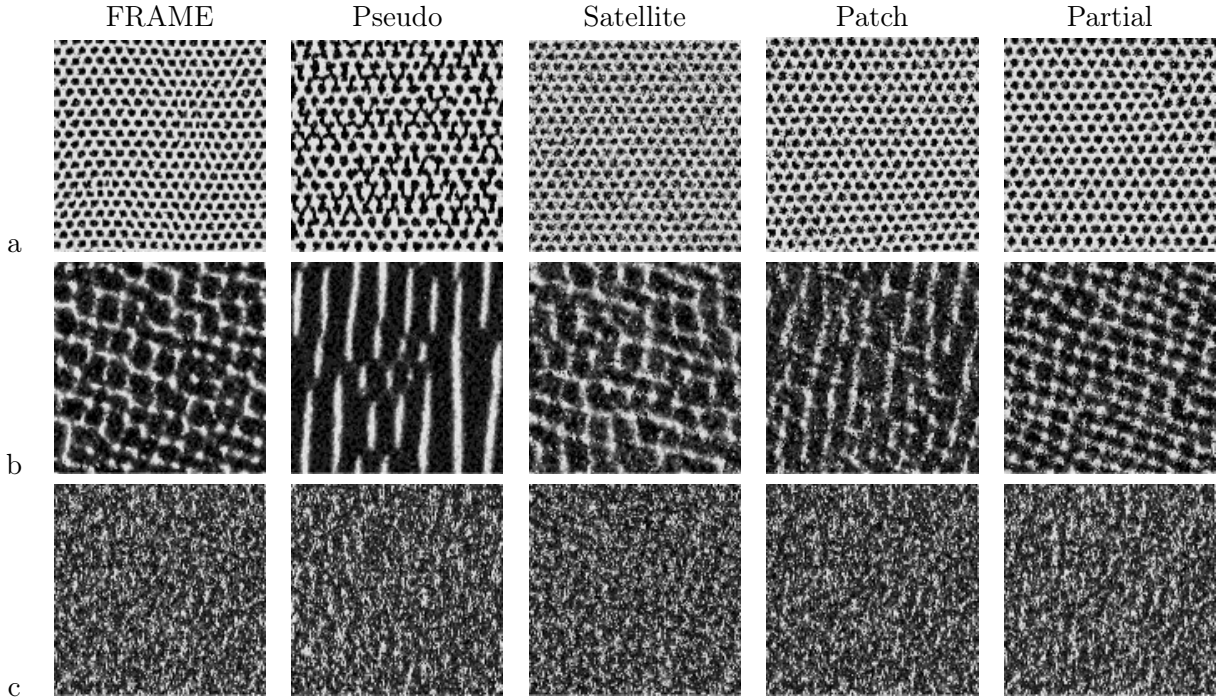


Figure 10.4: Synthesized texture images using β learned from various algorithms. For each column from left to right. 1: Stochastic gradient algorithm as the ground truth, 2: pseudo-likelihood, 3: satellite likelihood, 4: patch likelihood, 5: partial likelihood.

Since for different textures the model $p(\mathbf{I}; \beta)$ may be more sensitive to some elements of β (such as tail bins) than to the rest parameters, and the β vectors are highly correlated between its components, it is not very meaningful to compare the accuracy of the learned β using an error measure $|\beta - \beta^*|$. Instead we sample each learned model $\mathbf{I}^{\text{syn}} \sim p(\mathbf{I}; \beta)$ and compare the histogram errors of the synthesized image against the observed, i.e. $|\mathbf{h}(\mathbf{I}^{\text{syn}}) - \mathbf{h}(\mathbf{I}^{\text{obs}})|$, summed over 12 pairs of histograms each being normalized to 1. The table below shows the errors for each algorithms for the synthesized images in Figure 10.4. The numbers are subject to some computational fluctuations including the sampling process.

Fig.4.	FRAME	Pseudo	Satellite	Patch	Partial
Fig.4.a	(0.449)	(2.078)	(1.704)	(1.219)	(1.559)
Fig.4.b	(0.639)	(2.555)	(1.075)	(1.470)	(1.790)
Fig.4.c	(0.225)	(0.283)	(0.325)	(0.291)	(0.378)

The experimental results show that the four algorithms work reasonably well. In comparison the satellite method is often close to the patch and partial likelihood methods. Though it sometimes may yield slightly better results than other methods depending on the similarity between the reference models and the model to be learned. The pseudo-likelihood method can also capture some large image features. In particular, it works well for textures of stochastic nature. For example, on the texture in Figure 10.4.c.

In terms of computational complexity, the satellite algorithm is the fastest, and it computes β in the order of 10 seconds in a HP-workstation. The second fastest is the pseudo-likelihood. It takes a few minutes. However, the pseudo likelihood method consumes a large memory, as it needs to remember all the k histograms for the g grey levels in $N \times N$ pixels. The space complexity is $O(N^2 \times g \times k \times B)$ with B the number of bins. It often needs more than one Gigabyte of memory. The

partial likelihood and patch likelihood are very similar to the stochastic gradient algorithm [222]. Since the initial boundary condition is typical, these two estimators take only in general 1/10th of the number of sweeps to convergence. In addition, only a portion of pixels need to be synthesized, which can save computation further. The computation time is only about 1/20th of the stochastic gradient algorithm.

Experiment II: Analysis of the maximum satellite likelihood estimator.

In the second experiment, we study how the performance of the satellite algorithm is influenced by 1). The boundary condition, and 2). The size of patch $m \times m$.

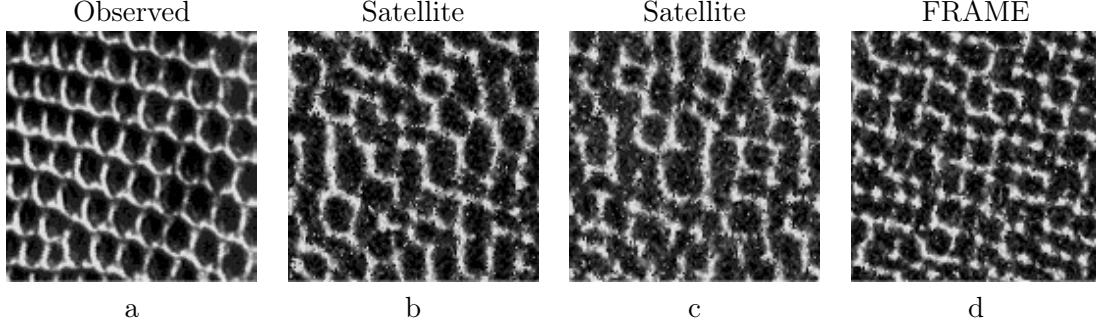


Figure 10.5: Performance evaluation of the satellite algorithm. (a) Observed texture image. (b) Synthesized image using β learned without boundary conditions. (c) Synthesized image using β learned with boundary conditions. (d) Synthesized image using β learned by stochastic gradient.

I). Influence of boundary conditions.

Figure 10.5.a displays a texture image as \mathbf{I}^{obs} . We run three algorithms for comparison. Figure 10.5.d is a result from the FRAME (stochastic gradient method). Figure 10.5.b and c are results using the satellite algorithms. The difference is that Figure 10.5.c uses observed boundary condition for each patch and does on-line sampling, while Figure 10.5.b ignores the boundary condition. For all the following results of satellite likelihood method (algorithm III), H_{ij}^{syn} are generated from the marginal probabilities without on-line sampling.

II). Influences of the hole size $m \times m$.

We fix the total number of foreground pixels $\sum_i |\Lambda_i|$ and study the performance of satellite algorithm with different hole sizes m . Figures 10.6.a-c show three synthesized images using β learned by satellite algorithm with different hole sizes $m = 2, 6, 9$ respectively. It is clear from Figures 10.6.a-c that the hole size with 6×6 pixels gives better result.

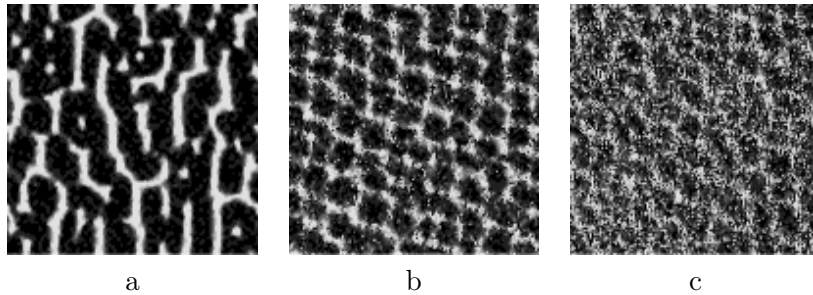


Figure 10.6: Synthesized images using β learned by the satellite method with different hole sizes. (a) $m = 2$. (b) $m = 6$. (c) $m = 9$.

To explain why the hole size of $m = 6$ gives better satellite approximation, we compute the two key factors that determine performance. Figure 10.7.a shows the numeric results in correspondence to the theoretical analysis displayed in Figure 10.2.

When the hole size is small, the partition function can be estimated accurately as shown by the

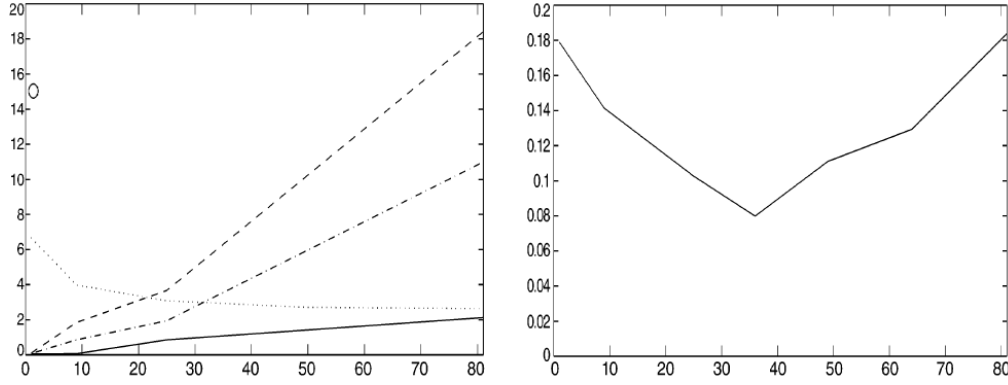


Figure 10.7: The x-axes are the hole size m^2 . a). Dotted curve is $E_f[(\hat{\beta} - \beta^*)^2]$ plotted against the hole size m^2 . The solid, dash-dotted, and dashed curves are $E_p[(\hat{Z} - Z)^2]$ for three different reference models. b). Average synthesis error per filter with respect to the hole size m^2 .

small $E_p[(\hat{Z} - Z)^2]$ in solid, dash-dotted, and dashed curves in Figure 10.7. However, the variance $E_f[(\hat{\beta} - \beta^*)^2]$ is large for small holes, which is shown by the dotted curve in figure 10.7.a. The optimal choice of the hole size is thus approximately the intersection of the two curves. Since the reference models that we used are close to the dash-dotted line shown in figure 10.7.a, we predict optimal hole size is between 5×5 and 6×6 . Figure 10.7.b shows the average error between the statistics of synthesized image $\mathbf{I}^{\text{syn}} \sim p(\mathbf{I}; \beta)$ and the observed statistics $\text{err} = \frac{1}{12} |\mathbf{h}(\mathbf{I}^{\text{obs}}) - \mathbf{h}(\mathbf{I}^{\text{syn}})|$, where β is learned using the satellite method for $m = 1, 2, \dots, 9$. Here the hole size of 6×6 pixels gives better result.

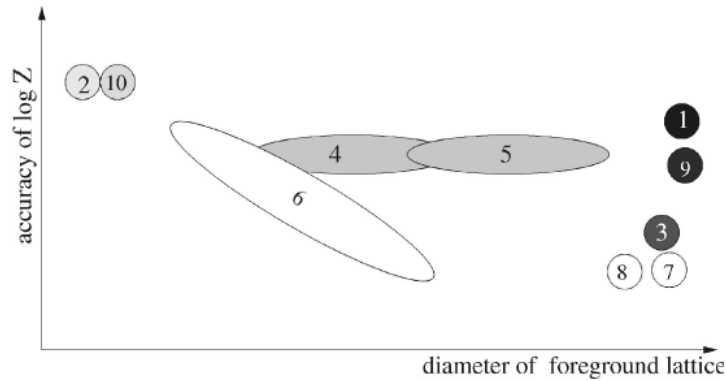


Figure 10.8: A common framework for learning Gibbs models. The horizontal axis is the size of foreground patches which is proportional to Fisher's information. The vertical axis is the accuracy in estimating $\log Z$. The brightness of the ellipses implies the learning speed, and darker is slower. This figure is intended only for a qualitative comparison.

Chapter 11

Mapping the Energy Landscape

In many statistical learning problems, the energy functions to be optimized are highly non-convex. A large body of research has been devoted to either approximating the target function by convex optimization, such as replacing L_0 norm by L_1 norm in regression, or designing algorithms to find a good local optimum, such as EM algorithm for clustering. Much less work has been done in analyzing the properties of such non-convex energy landscapes.

In this chapter, inspired by the success of visualizing the landscapes of Ising and Spin-glass models by [12] and [217], we compute *Energy Landscape Maps* (ELMs) in the high-dimensional model spaces (i.e. the hypothesis spaces in the machine learning literature) for some classic statistical learning problems — clustering and bi-clustering.

The ELM is a tree structure, as Figure 11.1 illustrates, in which each leaf node represents a local minimum and each non-leaf node represents the barrier between adjacent energy basins. The ELM characterizes the energy landscape with the following information.

- The number of local minima and their energy levels;
- The energy barriers between adjacent local minima and their energy levels; and
- The probability mass and volume of each local minimum (See Figure 11.3).

Such information is useful in the following tasks.

1. Analyzing the intrinsic difficulty (or complexity) of the optimization problems, for either inference or learning tasks. For example, in bi-clustering, we divide the problem into the *easy*, *hard*, and *impossible* regimes under different conditions.

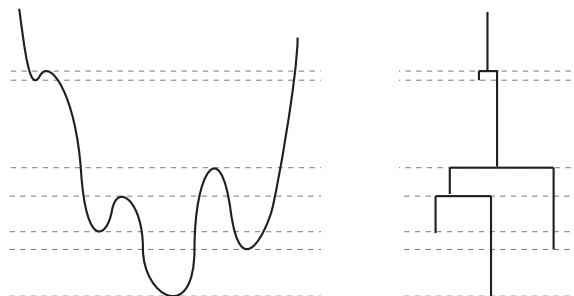


Figure 11.1: An energy function and the corresponding Energy Landscape Map (ELM). The y-axis of the ELM is the energy level, each leaf node is a local minimum and the leaf nodes are connected at the ridges of their energy basins.

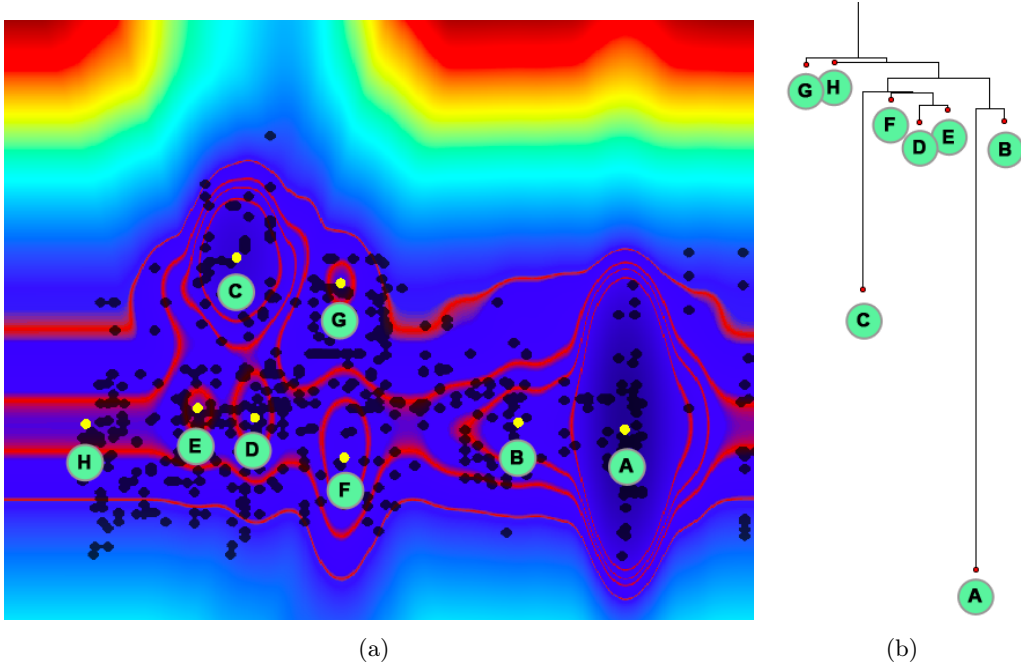


Figure 11.2: (a) Energy Landscape for a 4-component 1-d GMM with all parameters fixed except two means. Level sets are highlighted in red. The local minima are shown in yellow dots and the first 200 MCMC samples are shown in black dots. (b) The resulting ELM and the correspondence between the leaves and the local minima from the energy landscape.

2. Analyzing the effects of various conditions on the ELM complexity, for example, the separability in clustering, the number of training examples, the level of supervision (i.e. how many percent the examples are labeled), and the strength of regularization (i.e. prior model).
3. Analyzing the behavior of various algorithms by showing their frequencies of visiting the various minima. For example, in the multi-Gaussian clustering problem, we find that when the Gaussian components are highly separable, K-means clustering works better than the EM algorithm [51], and the opposite is true when the components are less separable. In contrast to the frequent visits of local minimum by K-means and EM, the Swendsen-Wang cut method [9] converges to the global minimum in all separability conditions.
4. Analyzing protein folding, where the energy landscape has a funnel-like appearance [154] with phase transitions. Before arriving at the bottom of the funnel, it is easy for the folding to move between different attraction basins, however the bottom could have a number of local optima of which only one is the native state (global optimum) and the others give rise to stable mis-folded proteins. Large quantities of such mis-folded proteins are related to neurodegenerative diseases such as Alzheimer's disease, Parkinson's, mad cow disease, etc.

We start with a simple illustrative example in Figures 11.2 and 11.3. Suppose the underlying probability distribution is a 4-component Gaussian mixture model (GMM) in 1D space, and the components are well separated. The model space is 11-dimensional with parameters $\{(\mu_i, \sigma_i, \alpha_i) : i = 1, 2, 3, 4\}$ denoting the means, variance and weights for each components. We sampled 70 data points from the GMM and construct the ELM in the model space. We bound the model space to a finite range defined by the samples.

As we can only visualize 2D maps, we set all parameters to equal the truth value except keeping μ_1 and μ_2 as the unknowns. Figure 11.2(a) shows the energy map on a range of $0 \leq \mu_1, \mu_2 \leq 5$. The

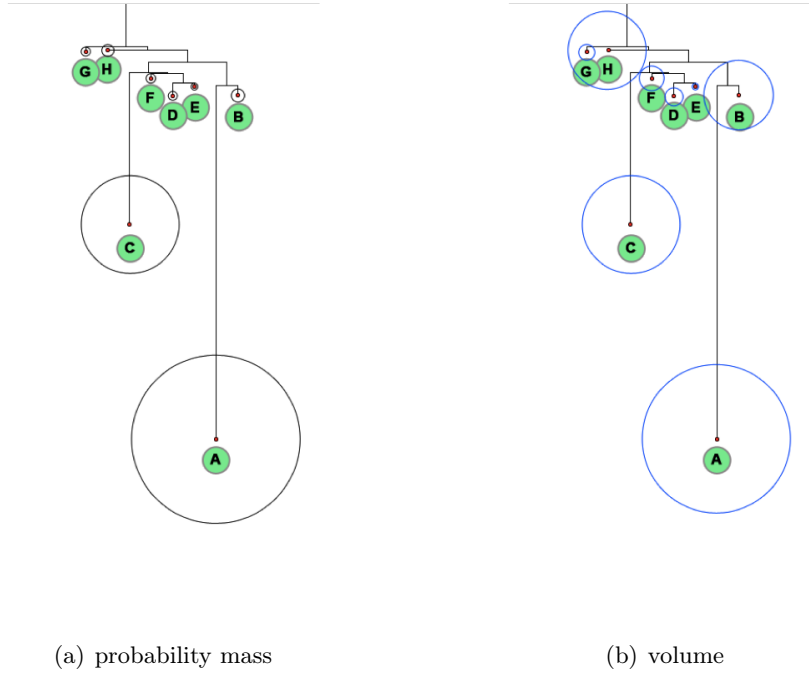


Figure 11.3: The probability mass and volume of the energy basins for the 2-d landscape shown in Figure 11.2.

asymmetry in the landscape is caused by the fact that the true model has different weights between the first and second component. Some shallow local minima, like E, F, G, H, are little “dents” caused by the finite data samples.

Figure 11.2 (a) shows that all the local minima are identified. Additionally, it shows the first 200 MCMC samples that were accepted by the algorithm that we will discuss later. The samples are clustered around the local minima, and cover all energy basins. They are not present in the high energy areas away from the local minima, as would be desired. Figure 11.2 (b) shows the resulting ELM and the correspondence between the leaves and the local minima in the energy landscape. Furthermore, Figures 11.3 (a) and (b) show the probability mass and the volume of these energy basins.

In the literature, [12] presents the first work for visualizing multidimensional energy landscapes for the spin-glass model. Since then statisticians have developed a series of MCMC methods for improving the efficiency of the sampling algorithms traversing the state spaces. Most notably, [118] generalizes the Wang-Landau algorithm [201] for random walks in the state space. [217] uses the generalized Wang-Landau algorithm to plot the disconnectivity graph for Ising model with hundreds of local minima and proposes an effective way for estimating the energy barriers. Furthermore, [218] construct the energy landscape for Bayesian inference of DNA sequence segmentation by clustering Monte Carlo samples.

In contrast to the above work that compute the landscapes in “state” spaces for inference problems, our work is focused on the landscapes in “model” spaces (the sets of all models; also called hypothesis spaces in the machine learning community) for statistical learning and model estimation problems. There are some new issues in plotting the model space landscapes. i) Many of the basins have a flat bottom, for example, basin A in Figure 11.2.(a). This may result in a large number of false local minima. ii) There may be constraints between parameters, for example, the weights have to sum up to one — $\sum_i \alpha_i = 1$. Thus we may need to run our algorithm on a manifold.

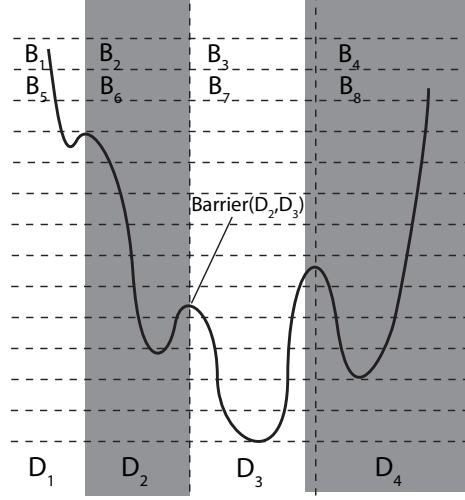


Figure 11.4: The model space Ω is partitioned into energy basins D_i (along the x-axis), and the energy \mathbb{R} (the y-axis) is partitioned into uniform intervals $[u_{j+1}, u_j]$.

11.1 ELM construction

In this section, we introduce the basic ideas for constructing the ELM and estimating its properties - mass, volume and complexity.

11.1.1 Space partition

Let Ω be the model space over which a probability distribution $\pi(x)$ and energy $E(x)$ are defined. In this paper, we assume Ω is bounded using properties of the samples. Ω is partitioned into K disjoint subspaces which represent the energy basins

$$\Omega = \cup_{i=1}^K D_i, \quad \cap_{i=1}^K D_i = \emptyset \quad \forall i \neq j. \quad (11.1)$$

That is, any point $x \in D_i$ will converge to the same minimum through gradient descent.

As Figure 11.4 shows, the energy is also partitioned into intervals $[u_{j+1}, u_j], j = 1, 2, \dots, L$. Thus we obtain a set of bins as the quantized atomic elements in the product space $\Omega \times \mathbb{R}$,

$$B_{ij} = \{x : x \in D_i, E(x) \in [u_{j+1}, u_j]\}. \quad (11.2)$$

The number of basins K and the number of intervals L are unknown and have to be estimated during the computing process in an adaptive and iterative manner.

11.1.2 Generalized Wang-Landau algorithm

The objective of the generalized Wang-Landau (GWL) algorithm is to simulate a Markov chain that visits all the bins $\{B_{ij}, \forall i, j\}$ with equal probability, and thus effectively reveal the structure of the landscape.

Let $\phi : \Omega \rightarrow \{1, \dots, K\} \times \{1, \dots, L\}$ be the mapping between the model space and bin indices: $\phi(x) = (i, j)$ if $x \in B_{ij}$. Given any x , by gradient descent or its variants, we can find and record the basin D_i that it belongs to, compute its energy level $E(x)$, and thus find the index $\phi(x)$.

We define $\beta(i, j)$ to be the probability mass of a bin

$$\beta(i, j) = \int_{B_{i,j}} \pi(x) dx. \quad (11.3)$$

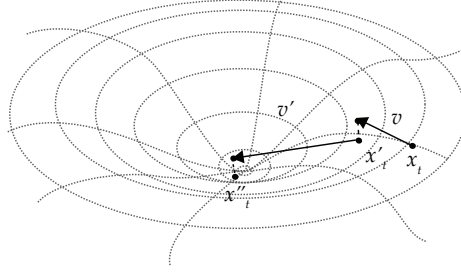


Figure 11.5: First two steps of projected gradient descent. The algorithm is initialized with MCMC sample x_t . v is the gradient of $E(x)$ at the point x_t . Armijo line search is used to determine the step size α along the vector v . x'_t is the projection $T(x_t + \alpha v)$ onto the subspace Γ . Then x''_t is the projection $T(x_t + \alpha' v')$, and so on.

Then, we can define a new probability distribution which has equal probability among all the bins,

$$\pi'(x) = \frac{1}{Z} \pi(x) / \beta(\phi(x)), \quad (11.4)$$

with Z being a scaling constant.

To sample from $\pi'(x)$, one can estimate $\beta(i, j)$ by a variable γ_{ij} . We define the probability function $\pi_\gamma : \Omega \rightarrow \mathbb{R}$ to be

$$\pi_\gamma(x) \propto \frac{\pi(x)}{\gamma_{\phi(x)}} = \sum_{i,j} \frac{\pi(x)}{\gamma_{ij}} \mathbb{1}(x \in B_{ij}) \quad \text{st.} \quad \int_{\Omega} \pi_\gamma(x) dx = 1.$$

We start with an initial γ^0 , and update $\gamma^t = \{\gamma_{ij}^t, \forall i, j\}$ iteratively using stochastic approximation [119]. Suppose x_t is the MCMC state at time t , then γ^t is updated in an exponential rate,

$$\log \gamma_{ij}^{t+1} = \log \gamma_{ij}^t + \eta_t \mathbb{1}(x_t \in B_{ij}), \quad \forall i, j. \quad (11.5)$$

η_t is the step size at time t . The step size is decreased over time and the decreasing schedule is either pre-determined as in [119] or determined adaptively as in [216].

Each iteration with given γ^t uses a Metropolis step. Let $Q(x, y)$ be the proposal probability for moving from x to y , then the acceptance probability is

$$\begin{aligned} \alpha(x, y) &= \min \left(1, \frac{Q(y, x) \pi_\gamma(y)}{Q(x, y) \pi_\gamma(x)} \right) \\ &= \min \left(1, \frac{Q(y, x)}{Q(x, y)} \frac{\pi(y)}{\pi(x)} \frac{\gamma_{\phi(x)}^t}{\gamma_{\phi(y)}^t} \right). \end{aligned} \quad (11.6)$$

Intuitively, if $\gamma_{\phi(x)}^t < \gamma_{\phi(y)}^t$, then the probability of visiting y is reduced. For the purpose of exploring the energy landscape, the GWL algorithm improves upon conventional methods, such as the simulated annealing [82] and tempering [134] process. The latter sample from $\pi(x)^{\frac{1}{T}}$ and do not visit the bins with equal probability even at high temperature.

In performing gradient descent, we employ Armijo line search to determine the step size; if the model space Ω is a manifold in \mathbb{R}^n , we perform projected gradient descent, as shown in Figure 11.5. To avoid erroneously identifying multiple local minima within the same basin (especially when there is large flat regions), we merge local minima identified by gradient descent based on the following criteria: (1) the distance between two local minima is smaller than a constant ϵ ; or (2) there is no barrier along the straight line between two local minima.

Figure 11.6 (a) illustrates a sequence of Markov chain states x_t, \dots, x_{t+9} over two energy basins. The dotted curves are the level sets of the energy function.

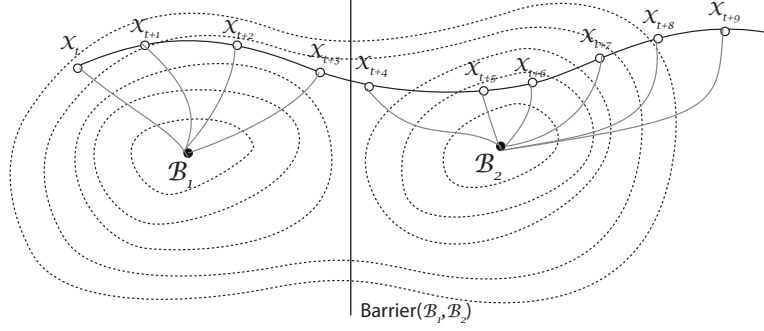


Figure 11.6: Sequential MCMC samples $x_t, x_{t+1}, \dots, x_{t+9}$. For each sample, we perform gradient descent to determine which energy basin the sample belongs to. If two sequential samples fall into different basins (x_{t+3} and x_{t+4} in this example), we estimate or update the upper-bound of the energy barrier between their respective basins (B_1 and B_2 in this example).

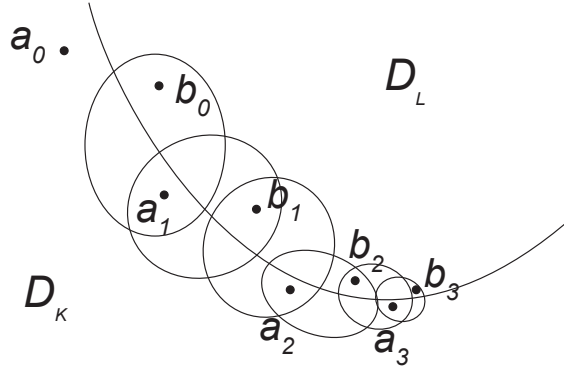


Figure 11.7: The ridge descent algorithm is used for estimating the energy barrier between basins D_k and D_l initialized at consecutive MCMC samples $a_0 = x_t, b_0 = x_{t+1}$ where $a_0 \in D_k$ and $b_0 \in D_l$.

11.1.3 Constructing the ELM

Suppose we have collected a chain of samples x_1, \dots, x_N from the GWL algorithm. The ELM construction consists of the following two processes.

1, *Finding the energy barriers between adjacent basins.* We collect all consecutive MCMC states that move across two basins D_k and D_l ,

$$X_{kl} = \{(x_t, x_{t+1}) : x_t \in D_k, x_{t+1} \in D_l\} \quad (11.7)$$

we choose $(a_0, b_0) \in X_{kl}$ with the lowest energy

$$(a_0, b_0) = \operatorname{argmin}_{(a,b) \in \Omega_{kl}} [\min(E(a), E(b))].$$

Next we iterate the following step as Figure 11.7 illustrates

$$\begin{aligned} a_i &= \operatorname{argmin}_a \{E(a) : a \in \text{Neighborhood}(b_{i-1}) \cap D_k\} \\ b_i &= \operatorname{argmin}_b \{E(b) : b \in \text{Neighborhood}(a_i) \cap D_l\} \end{aligned}$$

until $b_{i-1} = b_i$. The neighborhood is defined by an adaptive radius. Then b_i is the energy barrier and $E(b_i)$ is the energy level of the barrier. A discrete version of this ridge descent method was used in [217].

2, *Constructing the tree structure.* The tree structure of the ELM is constructed from the set of energy basins and the energy barriers between them via an iterative algorithm modified from the hierarchical agglomerative clustering algorithm. Initially, the energy basins are represented by leaf nodes that are not connected, whose y-coordinates are determined by the local minima of the basins. In each iteration, the two nodes representing the energy basins D_1, D_2 with the lowest barrier are connected by a new parent node, whose y-coordinates is the energy level of the barrier; D_1 and D_2 are then regarded as merged, and the energy barrier between the merged basin and any other basin D_i is simply the lower one of the energy barriers between D_1/D_2 and D_i . When all the energy basins are merged, we obtain the complete tree structure. For clarity, we can remove from the tree basins of depth less than a constant ϵ .

11.1.4 Estimating the mass and volume of nodes in the ELM

In the ELM, we can estimate the probability mass and the volume of each energy basin. When the algorithm converges, the normalized value of γ_{ij} approaches the probability mass of bin B_{ij} :

$$\hat{P}(B_{ij}) = \frac{\gamma_{ij}}{\sum_{kl} \gamma_{kl}} \rightarrow \beta(i, j), \quad \text{almost surely.}$$

Therefore the probability mass of a basin D_i can be estimated by

$$\hat{P}(D_i) = \sum_j \hat{P}(B_{ij}) = \frac{\sum_j \gamma_{ij}}{\sum_{kl} \gamma_{kl}} \quad (11.8)$$

Suppose the energy $E(x)$ is partitioned into sufficiently small intervals of size du . Based on the probability mass, we can then estimate the size¹ of the bins and basins in the model space Ω . A bin B_{ij} with energy interval $[u_j, u_j + du)$ can be seen as having energy u_j and probability density αe^{-u_j} (α is a normalization factor). The size of bin B_{ij} can be estimated by

$$\hat{A}(B_{ij}) = \frac{\hat{P}(B_{ij})}{\alpha e^{-u_j}} = \frac{\gamma_{ij}}{\alpha e^{-u_j} \sum_{kl} \gamma_{kl}}$$

The size of basin D_i can be estimated by

$$\hat{A}(D_i) = \sum_j \hat{A}(B_{ij}) = \frac{1}{\sum_{kl} \gamma_{kl}} \sum_j \frac{\gamma_{ij}}{\alpha e^{-u_j}} \quad (11.9)$$

Further, we can estimate the volume of a basin in the energy landscape which is defined as the amount of space contained in a basin in the space of $\Omega \times \mathbb{R}$.

$$\hat{V}(D_i) = \sum_j \sum_{k: u_k \leq u_j} \hat{A}(B_{ik}) \times du = \frac{du}{\sum_{lm} \gamma_{lm}} \sum_j \sum_{k: u_k \leq u_j} \frac{\gamma_{ik}}{\alpha e^{-u_k}} \quad (11.10)$$

where the range of j depends on the definition of the basin. In a restricted definition, the basin only includes the volume under the closest barrier, as Figure 11.8 illustrates. The volume above the basins 1 and 2 is shared by the two basins, and is between the two energy barriers C and D . Thus we define the volume for a non-leaf node in the ELM to be the sum of its children plus the volume between the barriers. For example, node C has volume $V(A) + V(B) + V(AB)$.

¹Note that the size of a bin/basin in the model space is called its volume by [218], but here we will use the term “volume” to denote the capacity of a basin in the energy landscape.

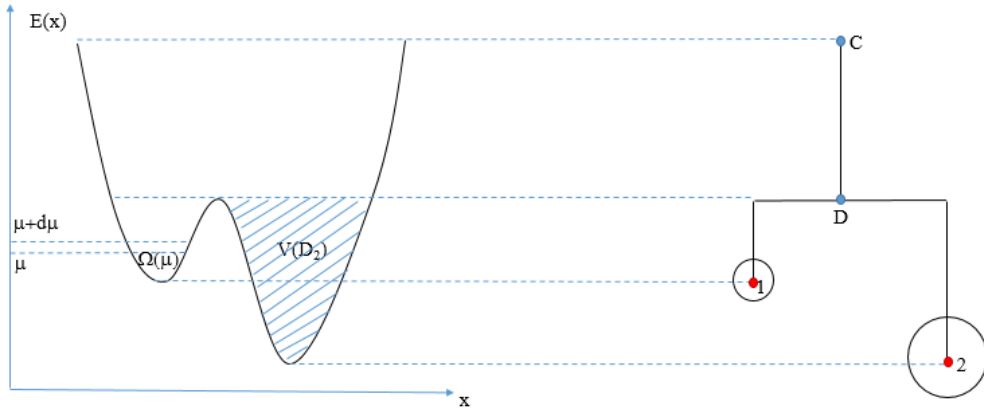


Figure 11.8: The volume of basins. Assuming that du is sufficiently small, the volume of an energy basin can be approximated by the summation of the estimated volume at each energy interval.

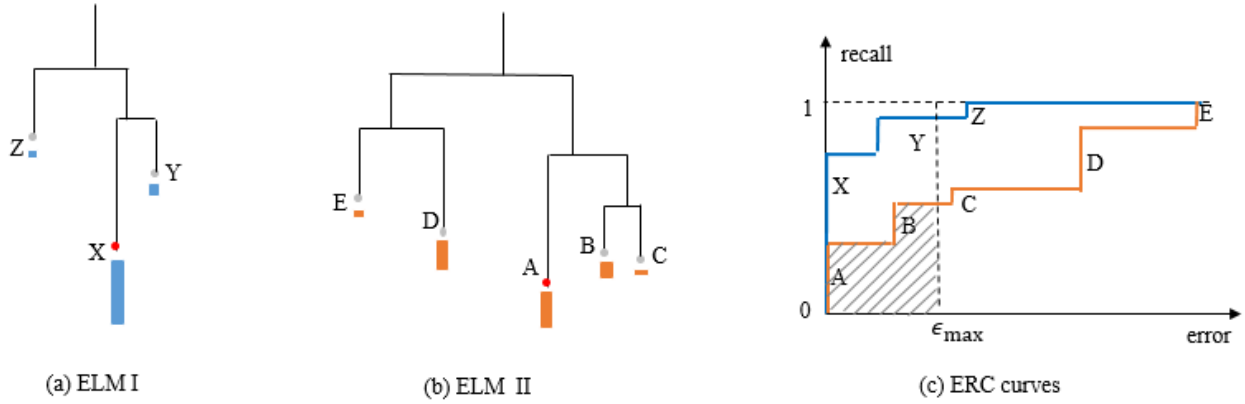


Figure 11.9: Characterizing the difficulty of learning in the ELM. For two learning tasks with ELM I and ELM II, the colored bar show the frequency that a learning algorithm converges to the basins, from which two Error-recall curves are plotted. The difficulty of learning task, with respect to this algorithm, can be measured by the area under the curve within an acceptable maximum error.

If our goal is to develop a scale-space representation of the ELM by repeatedly smoothing the landscape, then basins A and B will be merges into one basin at certain scale, and volume above the two basins will be also added to this new merged basin.

Note that the partition of the space into bins, rather than basins, facilitates the computation of energy barriers, the mass and volume of the basins.

11.1.5 Characterizing the difficulty (or complexity) of learning tasks

It is often desirable to measure the difficulty of the learning task by a single number. For example, we compare two ELMs in Figure 11.9. Learning in the landscape of ELM I looks easier than that of ELM II. However, the difficulty also depends on the learning algorithms. Thus we can run the learning algorithm many times and record the frequency that it converges to each basin or minimum. The frequency is shown by the lengths of the colored bars under the leaf nodes.

Suppose that Θ^* is the true model to be learned. In Figure 11.9, Θ^* corresponds to nodes X in ELM I and node A in ELM II. In general, Θ^* may not be the global minimum or not even a

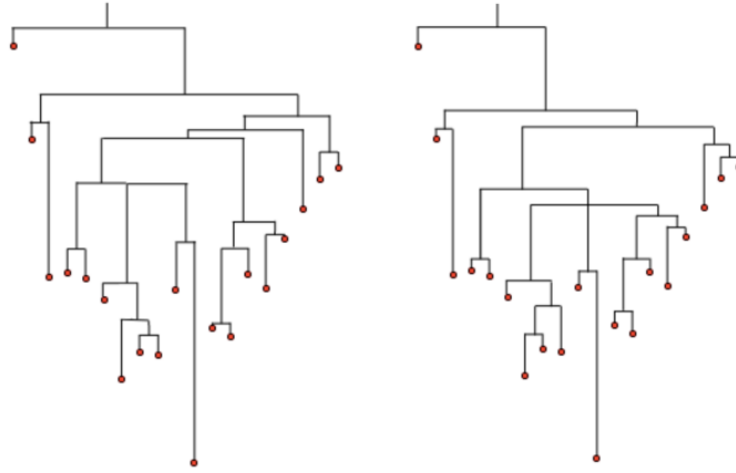


Figure 11.10: Two ELMs generated from two MCMC chains C_1 and C_2 initialized at different starting points after convergence in 24,000 iterations.

minimum. We then measure the distance (or error) between Θ^* and any other local minima. As the error increases, we accumulate the frequency to plot a curve. We call it the Error-Recall curve (ERC), as the horizontal axis is the error and the vertical axis is the frequency of recall the solutions. This is like the ROC (receptor-operator characteristics) curves in Bayesian decision theory, pattern recognition and machine learning. By sliding the threshold ϵ_{\max} which is maximum error tolerable, the curve characterizes the difficulty of the ELM with respect to the algorithm.

A single numeric number that characterizes the difficulty can be the area under the curve (AUC) for a given ϵ_{\max} . this is illustrated by the shadowed area in 11.9.(c) for ELM II. When AUC is close to 1, the task is easy, and when AUC is close to 0, learning is impossible.

In a learning problem, we can set different conditions which correspond to a range of ELMs. The difficulty measures of these ELMs can be visualized in the space of the parameters as a difficulty map. We will show such maps in experiment III.

11.1.6 MCMC moves in the model space

To design the Markov chain moves in the model space \mathbb{R} , we use two types of proposals in the metropolis-Hastings design in equation (11.6).

- 1, A random proposal probability $Q(x, y)$ in the neighborhood of the current model x .

- 2, Data augmentation. A significant portion of non-convex optimization problems involve latent variables. For example, in the clustering problem, the class label of each data point is latent. For such problems, we use data augmentation [181] to improve the efficiency of sampling. In order to propose a new model $y = x_{t+1}$, we first sample the values of the latent variables Z_t based on $p(Z_t|x_t)$ and then sample the new model x_{t+1} based on $p(x_{t+1}|Z_t)$. The proposal $y = x_{t+1}$ is then either accepted or rejected based on the same acceptance probability in Equation 11.6.

Note that, however, our goal in ELM construction is to traverse the model space instead of sampling from the probability distribution. When enough samples are collected and therefore the weights γ_{ij} become large, the reweighted probability distribution would be significantly different from the original distribution $\pi(x)$ and the rejection rate of the models proposed via data augmentation would become high. Therefore, we use the proposal probability based on data augmentation more often at the beginning and increasingly rely on random proposal when the weights become large.

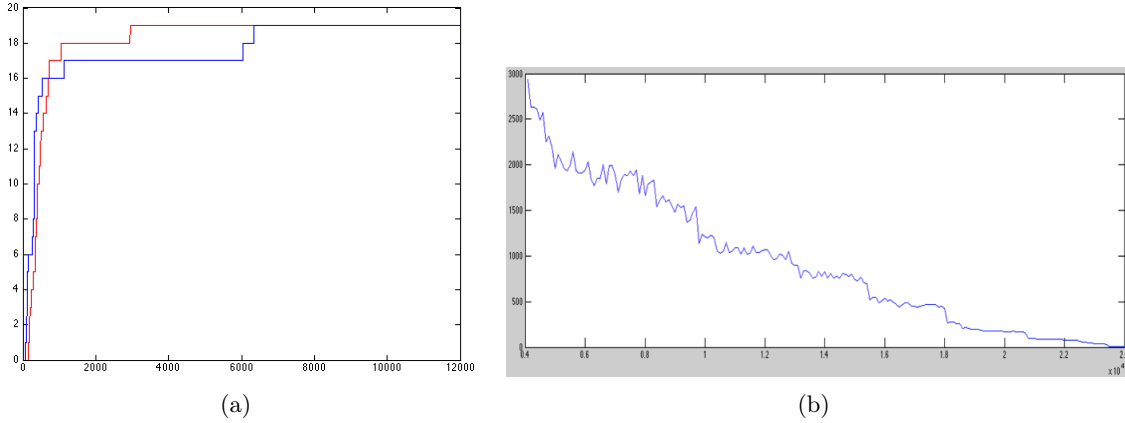


Figure 11.11: Monitoring the convergence of ELMs generated from two MCMC chains C_1 and C_2 initialized at different starting points. (a) The number of local minima found vs number of iterations for C_1 and C_2 . (b) the distance between the two ELMs vs. number of iterations.

11.1.7 ELM convergence analysis

The convergence of the GWL algorithm to a stationary distribution is a necessary but not sufficient condition for the convergence of the ELMs. As shown in Figure 11.10, the constructed ELMs may have minor variations due to two factors: (i) the left-right ambiguity when we plot the branches under a barrier; and (ii) the precision of the energy barriers will affect the internal structure of the tree.

In experiments, firstly we monitor the convergence of the GWL in the model space. We run multiple MCMC initialized with random starting values. After a burn-in period, we collect samples and project in a 2-3 dimensional space using Multi-dimensional scaling. We check whether the chains have converged to a stationary distribution using the multivariate extension of the Gelman and Rubin criterion [74] [27].

Once the GWL is believed to have converged, we can monitor the convergence of the ELM by checking the convergence of the following two sets over time t .

1. The set of leaf nodes of the tree S_L^t in which each point x is a local minimum with energy $E(x)$. As t increase, S_L^t grows monotonically until no more local minimum is found, as is shown in Figure 11.11.(a).
2. The set of internal nodes of the tree S_N^t in which each point y is an energy barrier at level $E(y)$. As t increases, we may find lower barrier as the Markov chain crosses different ridge between the basins. Thus $E(y)$ decreases monotonically until no barrier in S_N^t is updated during a certain time period.

We further calculate a distance measure between two ELMs constructed by two MCMCs with different initialization. To do so, we compute a best node matching between the two trees and then the distance is defined on the differences of the matched leaf nodes and barriers, and penalties on unmatched nodes. We omit the details of this definition as it is not important for this work. Figure 11.11.(b) shows the distance decreases as more samples are generated.

11.2 Experiment I: ELMs of Gaussian Mixture Models

In this section, we compute the ELMs for learning Gaussian mixture models for two purposes: (i) study the influences of different conditions, such as separability and level of supervision; and ii)

compare the behaviors and performances of popular algorithms including K-mean clustering, EM (Expectation-Maximization), two-step EM, and Swendsen-Wang cut. We will use both synthetic data and real data in the experiments.

11.2.1 Energy and Gradient Computations

A Gaussian mixture model Θ with n components in d dimensions have weights α_i , means μ_i and covariance matrices Σ_i for $i = 1, \dots, n$. Given a set of observed data points $\{z_i, i = 1, \dots, m\}$, we write the energy function as

$$E(\Theta) = -\log P(z_i : i = 1 \dots m | \Theta) - \log P(\Theta) \quad (11.11)$$

$$= -\sum_{i=1}^m \log f(z_i | \Theta) - \log P(\Theta). \quad (11.12)$$

$P(\Theta)$ is the product of a Dirichlet prior and a NIW prior. Its partial derivatives are trivial to compute. $f(z_i | \Theta) = \sum_{j=1}^n \alpha_j G(z_i; \mu_j, \Sigma_j)$ is the likelihood for data z_i , where $G(z_i; \mu_j, \Sigma_j) = \frac{1}{\sqrt{\det(2\pi\Sigma_j)}} \exp\left[-\frac{1}{2}(z_i - \mu_j)^T \Sigma_j^{-1}(z_i - \mu_j)\right]$ is a Gaussian model. In case a data point is labeled (i.e., from which component it is sampled is known), the likelihood is simply $G(z_i; \mu_j, \Sigma_j)$.

For a sample z_i , we have the following partial derivatives of the log likelihood for calculating the gradient in the energy landscape.

a), Partial derivative with respect to each weight α_j :

$$\frac{\delta \log f(z_i)}{\delta \alpha_j} = \frac{G(z_i; \mu_j, \Sigma_j)}{\sum_{k=1}^K \alpha_k G(z_i, \mu_k, \Sigma_k)}.$$

b), Partial derivative with respect to each mean μ_j :

$$\frac{\delta \log f(z_i)}{\delta \mu_j} = \frac{\alpha_j G(z_i; \mu_j, \Sigma_j)}{\sum_{k=1}^K \alpha_k G(z_i; \mu_k, \Sigma_k)} \Sigma_j^{-1}(\mu_j - z_i).$$

c), Partial derivative with respect to each covariance Σ_j :

$$\begin{aligned} \frac{\delta \log f_{\text{mm}}(z_i)}{\delta \Sigma_j} &= \frac{\alpha_j G(z_i; \mu_j, \Sigma_j)}{\sum_{k=1}^K \alpha_k G(z_i; \mu_k, \Sigma_k)} \frac{1}{2} \left[\frac{\delta}{\delta \Sigma_j} \log \alpha_j G(z_i; \mu_j, \Sigma_j) \right] \\ &= \frac{\alpha_j G(z_i; \mu_j, \Sigma_j)}{\sum_{k=1}^K \alpha_k G(z_i; \mu_k, \Sigma_k)} \frac{1}{2} \left[-\Sigma_j^{-T} + \Sigma_j^{-T} (z_i - \mu_j) (z_i - \mu_j)^T \Sigma_j^{-T} \right] \end{aligned}$$

During the computation, we need to restrict the Σ_j matrices so that each inverse Σ_j^{-1} exists in order to have a defined gradient. Each Σ_j is semi-positive definite, so each eigenvalue is greater than or equal to zero. Consequently we only need the minor restriction that for each eigenvalue λ_i of Σ_j , $\lambda_i > \epsilon$ for some $\epsilon > 0$. However, it is possible that after one gradient descent step, the new GMM parameters will be outside of the valid GMM space, i.e. the new Σ_j^{t+1} matrices at step $t+1$ will not be symmetric positive definite. Therefore, we need to project each Σ_j^{t+1} into the symmetric positive definite space with the projection

$$P_{\text{symm}}(P_{\text{pos}}(\Sigma_j^{t+1})).$$

The function $P_{\text{symm}}(\Sigma)$ projects the matrix into the space of symmetric matrices by

$$P_{\text{symm}}(\Sigma) = \frac{1}{2}(\Sigma + (\Sigma)^T).$$

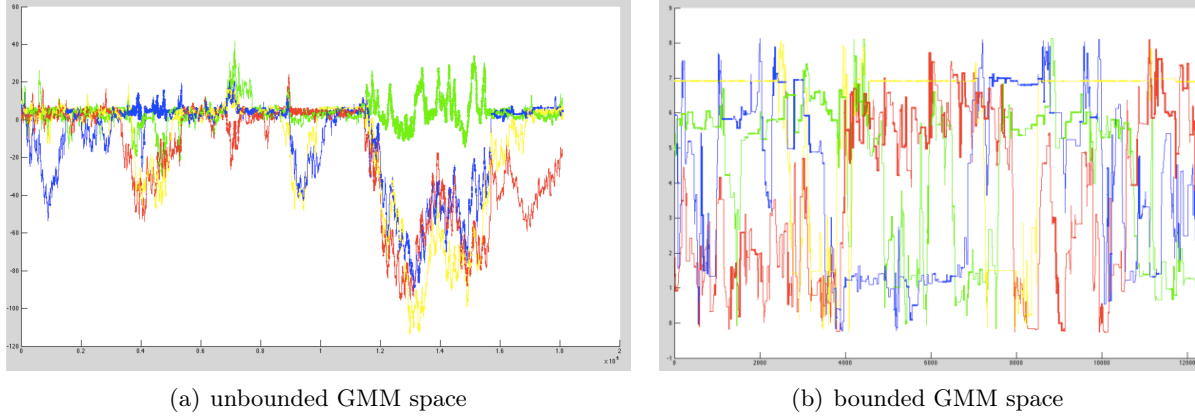


Figure 11.12: We sampled 70 data points from a 1-dimensional 4-component GMM and ran the MCMC random walk for ELM construction algorithm in the (a) unbounded (b) bounded GMM space. The plots show the evolution of the location of the centers of the 4 components over time. The width of the line represents the weight of the corresponding component.

Assuming that Σ is symmetric, the function $P_{\text{pos}}(\Sigma)$ projects Σ into the space of symmetric matrices with eigenvalues greater than ϵ . Because Σ is symmetric, it can be decomposed into $\Sigma = Q\Lambda Q^T$ where Λ is the diagonal eigenvalue matrix $\Lambda = \text{diag}\{\lambda_1, \dots, \lambda_n\}$, and Q is an orthonormal eigenvector matrix. Then the function

$$P_{\text{pos}}(\Sigma) = Q \begin{pmatrix} \max(\lambda_1, \epsilon) & 0 & \dots & 0 \\ 0 & \max(\lambda_2, \epsilon) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \max(\lambda_n, \epsilon) \end{pmatrix} Q^T$$

ensures that $P_{\text{pos}}(\Sigma)$ is symmetric positive definite.

11.2.2 Bounding the GMM space

From the m data points $\{z_i, i = 1, \dots, m\}$, we can estimate a boundary of the space of possible parameter Θ if m is sufficiently large.

Let μ_o and Σ_o be the sample mean and sample covariance matrix of all m points. We set a range for the means μ_j of the Gaussian components,

$$\|\mu_j - \mu_o\|_2 < \max_i \|z_i - \mu_o\|_2 + \epsilon_m.$$

ϵ_m is a constant that we will select in experiments. To bound the covariance matrices Σ_j , let $\Sigma_o = Q\Lambda Q^T$ be the eigenvalue decomposition of Σ_o with $\Lambda = \text{diag}\{\lambda_1, \dots, \lambda_n\}$. We denote by $L = \max(\lambda_1, \dots, \lambda_n) + \epsilon_m$ the upper bound of the eigen-values, and bound all the eigenvalues of Σ_j by L .

Figure 11.12 (a,b) compare the MCMCs in unbounded and bounded spaces respectively. We sampled $m = 70$ data points from a 1-dimensional, 4-component GMM and ran the MCMC random walk for ELM construction algorithm. The plots show the evolution of the locations of μ_1, \dots, μ_4 over time. Notice that in Figure 11.12 (a), the MCMC chain can move far from the center and spends the majority of the time outside of the bounded subspace. In Figure 11.12 (b), by forcing the chain to stay within the boundary, we are able to explore the relevant subspace more efficiently.

11.2.3 Experiments on Synthetic Data

We start with synthetic data with $n = 3$ component GMM on 2 dimensional space, draw m samples and run our algorithm to plot the ELM under different settings.

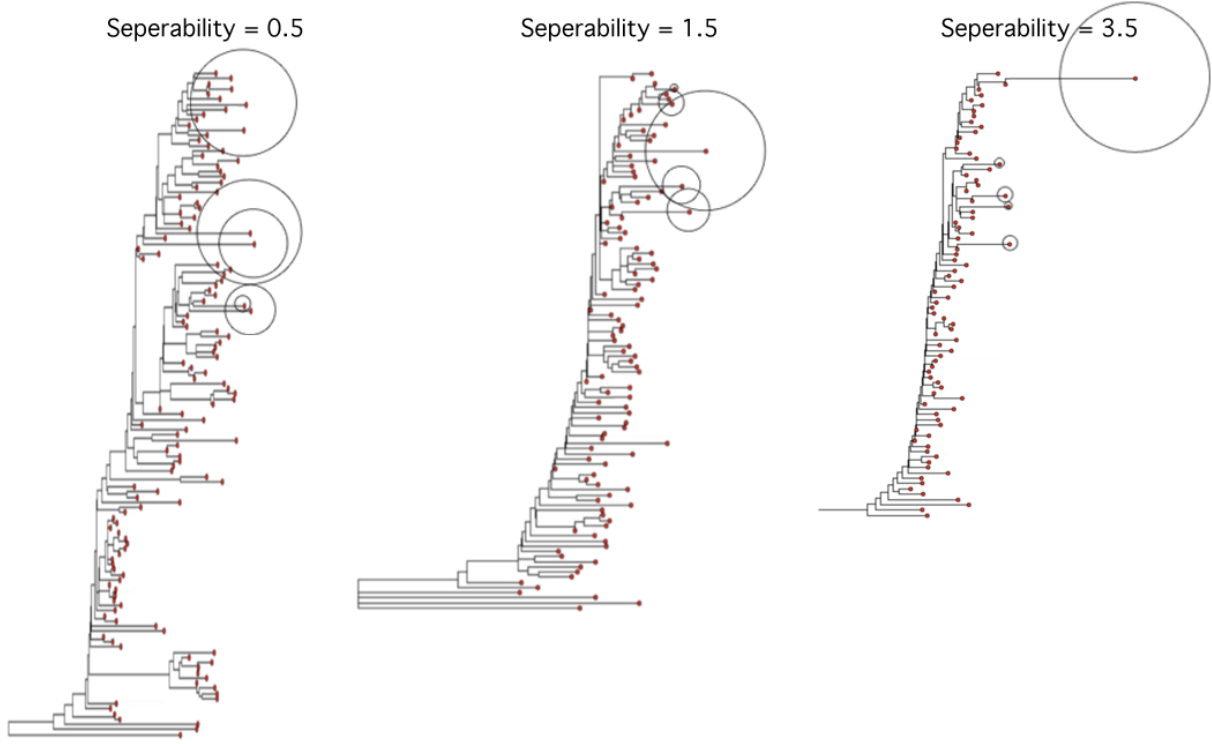


Figure 11.13: ELMs for $m = 100$ samples drawn from GMMs with low, medium and high separability $c = 0.5, 1.5, 3.5$ respectively. The circle represents the probability mass of the basins.

1) The effects of separability. The separability of the GMM represents the overlap between components of the model and is defined as $c = \min \left(\frac{\|\mu_i - \mu_j\|}{\sqrt{n} \max(\sigma_1, \sigma_2)} \right)$. This is often used in the literature to measure the difficulty of learning the true GMM model.

Figure 11.13 shows three representative ELMs with the separability $c = 0.5, 1.5, 3.5$ respectively for $m = 100$ data points. This clearly shows that at $c = 0.5$, the model is hardly identifiable with many local minima reaching similar energy levels. The energy landscape becomes increasingly simple as the separability increases. When $c = 3.5$, the prominent global minimum dominates the landscape.

2) The effects of partial supervision. We assign ground truth labels to a portion of the m data points. For z_i , its label ℓ_i indicates which component it belongs to. We set $m = 100$, separability $c = 1.0$. Figure 11.14 shows the ELMs with 0%, 5%, 10%, 50%, 90% data points labels. While unsupervised learning (0%) is very challenging, it becomes much simpler when 5% or 10% data are labeled. When 90% data are labeled, the ELM has only one minimum. Figure 11.15 shows the number of local minima in the ELM when labeling $1, \dots, 100$ samples. This shows a significant decrease in landscape complexity for the first 10\$ labels, and diminishing returns from supervised input after the initial 10%.

3) Behavior of Learning Algorithms. We compare the behaviors of the following algorithms under different separability conditions.

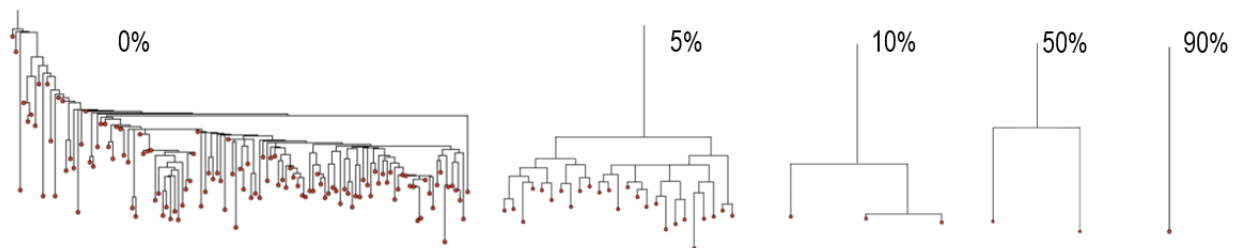


Figure 11.14: ELMS with of synthesized GMMs (separability $c = 1.0$, $n_{\text{Samples}} = 100$) with $\{0\%, 5\%, 10\%, 50\%, 90\%\}$ labelled data points.

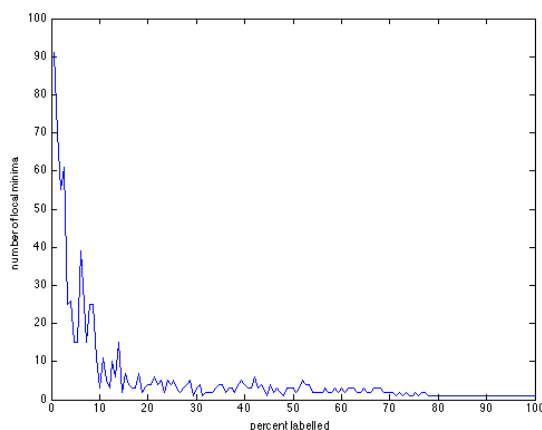


Figure 11.15: Number of local minima versus the percentage of labelled data points for a GMM with separability $c = 1.0$.

- Expectation-maximization (EM) is the most popular algorithms for learning GMM in statistics.
- K-means clustering is a popular algorithm in machine learning and pattern recognition.
- Two-step EM is a variant of EM proposed in [46] who have proved a performance guarantee under certain separability conditions. It starts with an excessive number of components and then prune them.
- The Swendsen-Wang Cut (SW-cut) algorithm proposed in [9]. This generalizes the SW method [180] from Ising/Potts models to arbitrary probabilities.

We modified EM, two-step EM and SW-cut in our experiments so that they minimize the energy function defined in Equation 11.11. K-means does not optimize our energy function, but it is frequently used as an approximate algorithm for learning GMM and therefore we include it in our comparison.

For each synthetic dataset in the experiment, we first construct the ELM, and then ran each of the algorithms for 200 times and record which of the energy basins the algorithm lands to. Hence we obtain the visiting frequency of the basins by each algorithm, which are shown as bars of varying length at the leaf nodes in Figures 11.16 and 11.17.

Figure 11.16 shows a comparison between the K-means, EM and two-step EM algorithms for $n = 10$ samples drawn from a low ($c = 0.5$) separability GMM. The results are scattered across

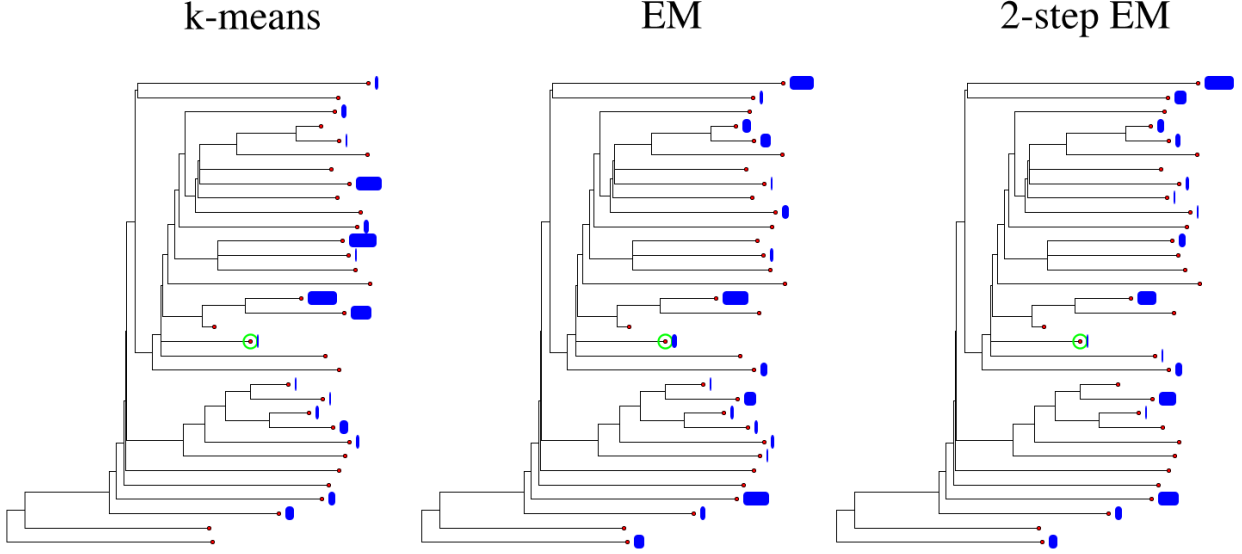


Figure 11.16: The performance of the k-means, EM and 2-step EM algorithms on the ELMs with 10 samples drawn from a GMM with low separability ($c = 0.5$)

different local minima regardless of the algorithm. This illustrates the difficulty in learning a model from a landscape with many local minima separated by large energy barriers.

Figure 11.17 show a comparison of the EM, k-means, and SW-cut algorithms for $m = 100$ samples drawn from low ($c = 0.5$) and high ($c = 3.5$) separability GMMs. The SW-cut algorithm performs best in each situation, always converging to the global optimal solution. In the low separability case, the k-means algorithm is quite random, while the EM algorithm almost always finds the global minimum and thus outperforms k-means. However, in the high separability case, the k-means algorithm converges to the true model the majority of the time, while the EM almost always converges to a local minimum with higher energy than the true model. This result confirms a recent theoretical result showing that the objective function of hard-EM (with k-means as a special case) contains an inductive bias in favor of high-separability models [172, 190]. Specifically, we can show that the actual energy function of hard-EM is:

$$E(\Theta) = -\log P(\Theta|Z) + \min_q (\mathbf{KL}(q(L)||P(L|Z, \Theta)) + H_q(L))$$

where Θ is the model parameters, $Z = z_1, \dots, z_m$ is the set of observable data points, L is the set of latent variables (the data point labels in a GMM), q is an auxiliary distribution of L , and H_q is the entropy of L measured with $q(L)$. The first term in the above formula is the standard energy function of clustering with GMM. The second term is called a posterior regularization term [71], which essentially encourages the distribution $P(L|Z, \Theta)$ to have a low entropy. In the case of GMM, it is easy to see that a low entropy in $P(L|Z, \Theta)$ implies high separability between Gaussian components.

11.2.4 Experiments on Real Data

We ran our algorithm to plot the ELM for the well-known Iris data set from the UCI repository [19]. The Iris data set contains 150 points in 4 dimensions and can be modeled by a 3-components GMM. The three components each represent a type of iris plant and the true component labels are known. The points corresponding to the first component are linearly separable from the others, but the points corresponding to the remaining two components are not linearly separable.

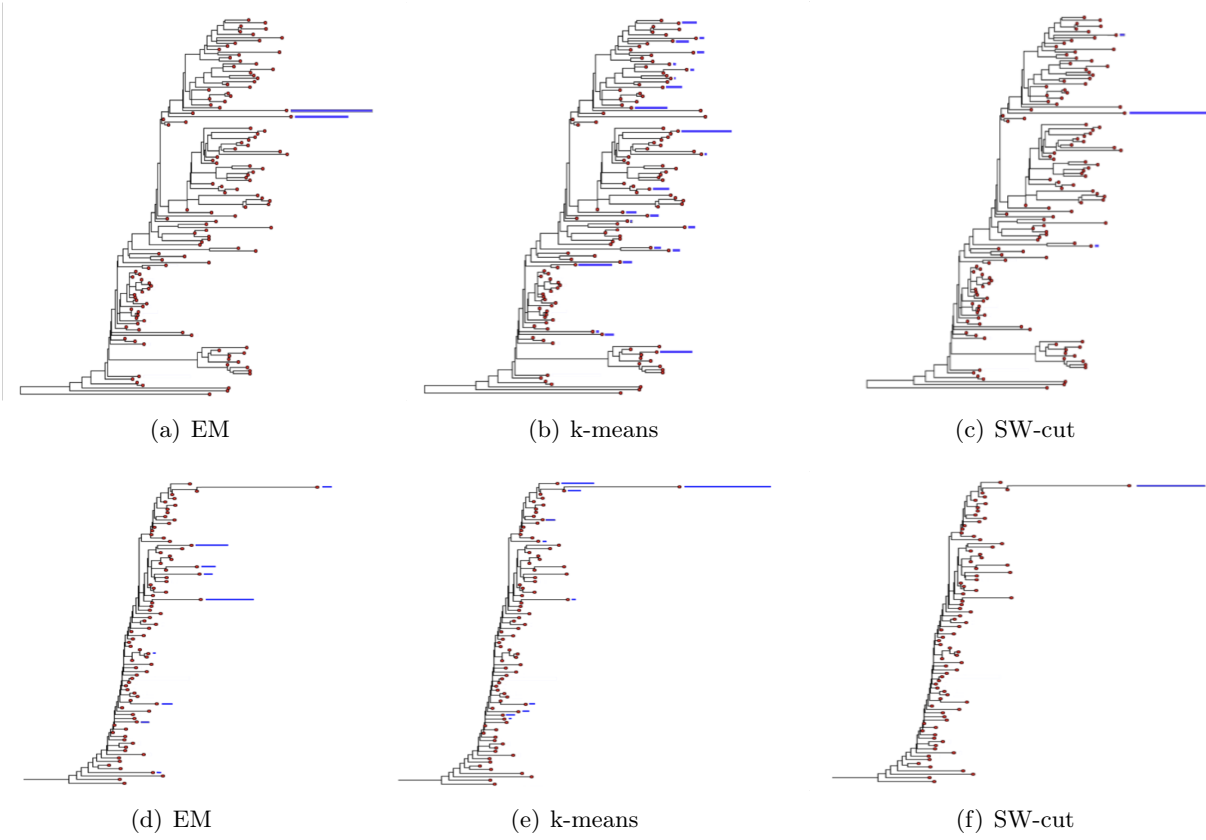


Figure 11.17: The performance of the EM, k-means, and SW-cut algorithm on the ELM. (a-c) Low separability $c = 0.5$. (d-f) High separability $c = 3.5$.

Figure 11.18 shows the ELM of the Iris dataset. We visualize the local minima by plotting the ellipsoids of the covariance matrices centered at the means of each component in 2 of the 4 dimensions.

The 6 lowest energy local minima are shown on the right and the 6 highest energy local minima are shown on the left. The high energy local minima are less accurate models than the low energy local minima. The local minima (E) (B) and (D) have the first component split into two and the remaining two (non-separable) components merged into one. The local minima (A) and (F) have significant overlap between the 2nd and 3rd components and (C) has the components overlapping completely. The low-energy local minima (G-L) all have the same 1st components and slightly different positions of the 2nd and 3rd components.

We ran the algorithm with 0, 5, 10, 50, 90, 100 percent of the points with the ground truth labels assigned. Figure 11.19 shows the global minimum of the energy landscape for these cases.

11.3 Experiment II: ELM of Bernoulli Templates

The synthetic data and Iris data in experiment I are in low dimensional spaces. In this section, we experiment with very high dimensional data for a learning task in computer vision and pattern recognition.

The objective is to learn a number of templates $BT_k, k = 1, \dots, K$ for object recognition. Figure 11.20 illustrates 10 templates of animal faces. Each template consists of a number of sketches or edges in the image lattice, and is denoted by a Boolean vector $BT_k = (s_{k1}, s_{k2}, \dots, s_{kn})$ with n

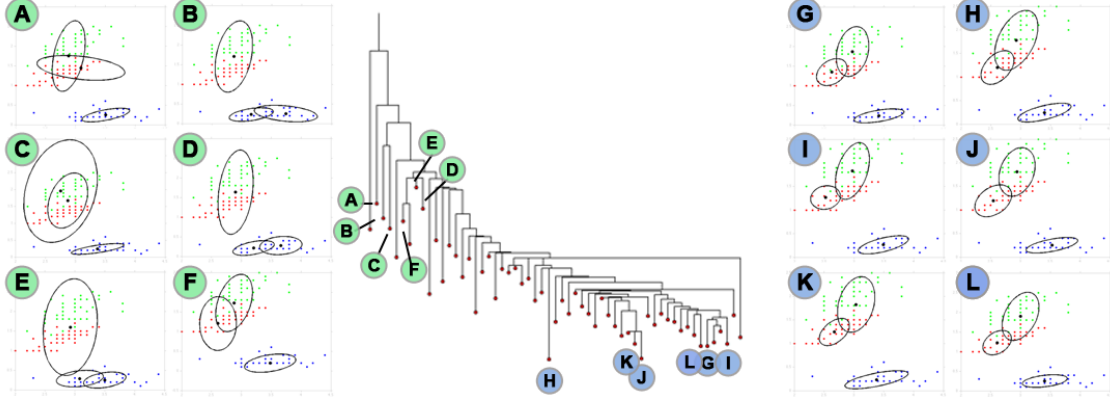


Figure 11.18: ELM and some of the local minima of the Iris dataset.

being the number of quantized positions and orientations of the lattice which is typically a large number $100 \sim 1000$. $s_{kj} = 1$ if there is a sketch at location j , and $s_{kj} = 0$ otherwise. Images are generated from one of the K templates with noise. Suppose $z_i = (r_{i1}, r_{i2}, \dots, r_{in})$ is an image generated from template BT_k , then $r_{ij} = s_{kj}$ with probability p and $r_{ij} = 1 - s_{kj}$ with probability $1 - p$. Thus we call $BT_k, k = 1, 2, \dots, K$ the Bernoulli templates. For simplicity we assume p is fixed for all the templates and all the locations.

The energy function that we use is the negative log of the posterior, given by $E(\Theta) = -\log P(\Theta|z_i : i = 1 \dots m)$ for m examples $\{z_i\}_{i=1}^m$. The model parameter Θ consists of the Boolean vectors $BT_k = (s_{k1}, s_{k2}, \dots, s_{kn})$ and the mixture weights α_k for $k = 1, \dots, K$. By assuming a uniform prior we have

$$P(\Theta|z_i : i = 1, \dots, m) = \prod_{i=1}^m \sum_{k=1}^K \alpha_k p^{\sum_{j=1}^n 1(r_{ij}=s_{kj})} (1-p)^{\sum_{j=1}^n 1(r_{ij} \neq s_{kj})},$$

In the following we present experiments on synthetic and real data.

11.3.1 Experiments on Synthetic Data

[7] proposes a Two-Round EM algorithm for learning Bernoulli templates with a performance bound that is dependent on the number of components K , the Bernoulli template dimension n , and noise level p . In this experiment, we examine how the ELM of the model space changes with these factors. We discretize the model space by allowing the weights to take values $\alpha_i \in \{0, 0.1, \dots, 1.0\}$. In order to adapt the GWL algorithm to the discrete space, we use coordinate descent in lieu of gradient descent.

We construct 10 Bernoulli templates which represent animal faces in Figure 11.20. Each animal face is aligned to a grid of 9×9 cells. Each cell may contain up to 3 sketches. Within a cell, the sketches are quantized to 18 discrete location and orientations. More specifically, each sketch is a straight line connecting the endpoints or midpoints of the edges of a square cell, and the 18 possible sketches in a cell are shown in Figure 11.21.(a). They can well approximate the detected edges or Gabor sketches from real images. The Bernoulli template can therefore be represented as a $n = 9 \times 9 \times 18$ dimensional binary vector. Figure 11.21.(b) shows a noisy dog face generated with noisy level $p = 0.1$.

We compute the ELMs of the Bernoulli mixture model for varying numbers of samples $m = 100, 300, \dots, 7000$ and varying noise level $p = 0, 0.05, \dots, 0.5, 0.55$. The number of local minima in each energy landscape is tabulated in Figure 11.22 (b) and drawn as a heat map in Figure 11.22 (a). As expected, the number of local minima increases as the noise level p increases, and decreases

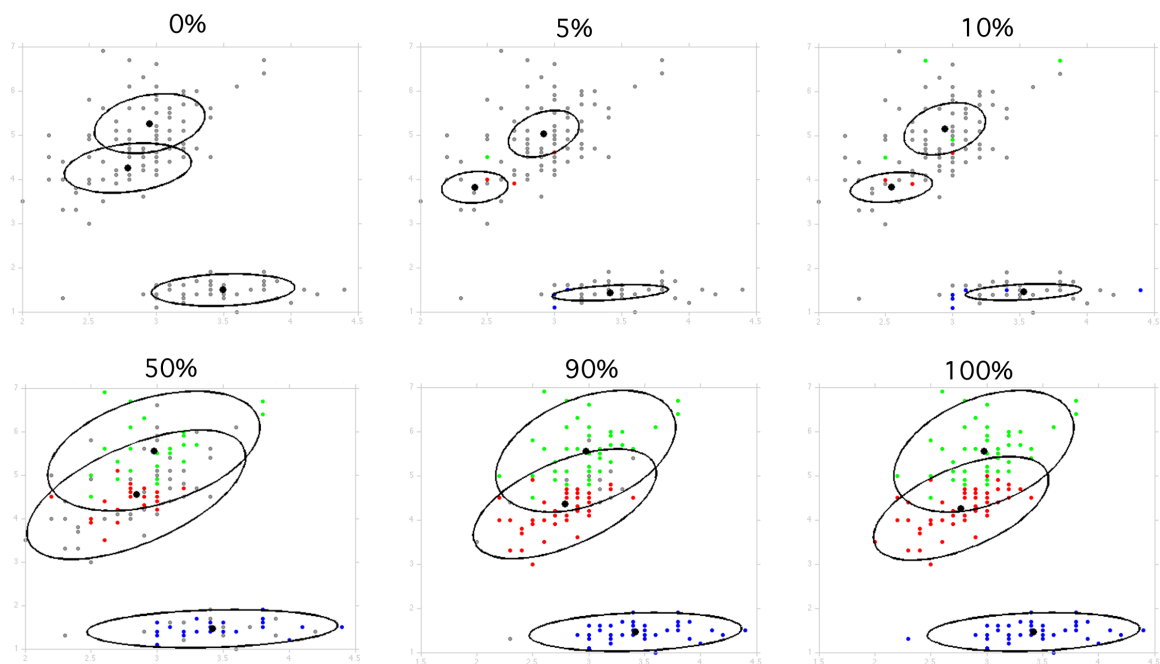


Figure 11.19: Global minima for learning from the Iris dataset with 0, 5, 10, 50, 90, and 100% of the data labeled with the ground truth values. Unlabeled points are drawn in grey and labelled points are colorized in red, green or blue.

as the number of samples decreases. In particular, with no noise, the landscape is convex and with noise $p > 0.45$, there are too many local minima and the algorithm does not converge.

We repeat the same experiment using variants of a mouse face. We swap out each component of the mouse face (the eyes, ears, whiskers, nose, mouth, head top and head sides) with three different variants. We thereby generate 20 Bernoulli templates in Figure 11.23, which have relatively high degrees of overlap. We generated the ELMs of various Bernoulli mixture models containing three of the 20 templates and noise level $p = 0$. In each Bernoulli mixture model, the three templates have different degrees of overlap. Hence we plot the number of local minima in the ELMs versus the degree of overlap as show in Figure 11.24. As expected, the number of local minima increases with the degree of overlap, and there are too many local minima for the algorithm to converge past overlap $c = 0.5$.

11.3.2 Experiments on Real Data

We perform the Bernoulli templates experiment on a set of real images of animal faces. We binarize the images by extracting the prominent sketches on a 9×9 grid. Eight Gabor filters with eight different orientations centered in the centers and corners of each cell are applied to the image. The filters with a strong response above a fixed threshold correspond to edges detected in the image; these are mapped to the dictionary of 18 elements. Thus each animal face is represented as a $18 \times 9 \times 9$ dimensional binary vector. The Gabor filter responses on animal face pictures are shown in Figure 11.25. The binarized animal faces are shown in Figure 11.26.

We chose 3 different animal types – deer, cat and mouse, with an equal number of images chosen from each category (Figure 11.27). The binarized versions of these images can be modeled as a mixture of 3 Bernoulli templates - each template corresponding to one animal face type.

The ELM is shown in Figure 11.28 along with the Bernoulli templates corresponding to three local minima separated by large energy barriers. We make two observations: 1. The templates corresponding to each animal type are clearly identifiable, and therefore the algorithm has converged

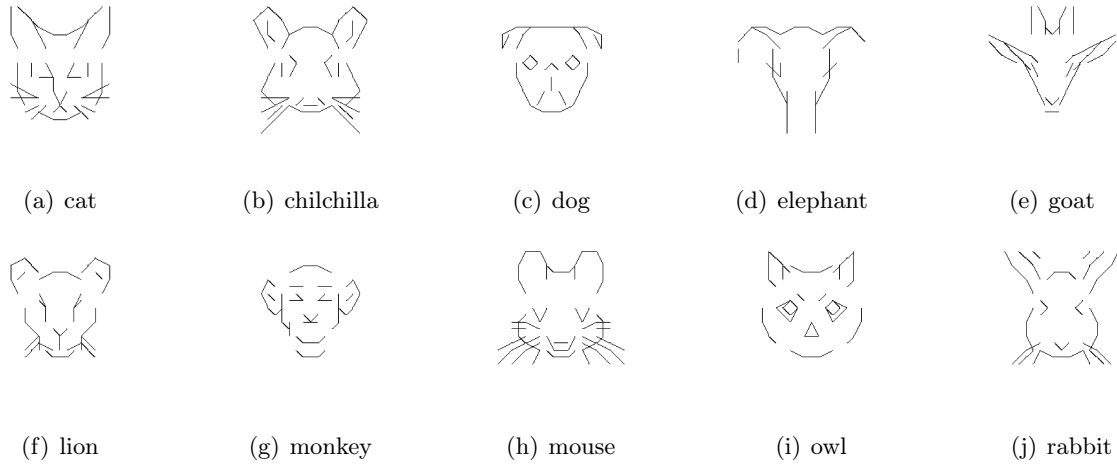


Figure 11.20: Bernoulli templates for animal faces. These templates have low overlap and are well separable.

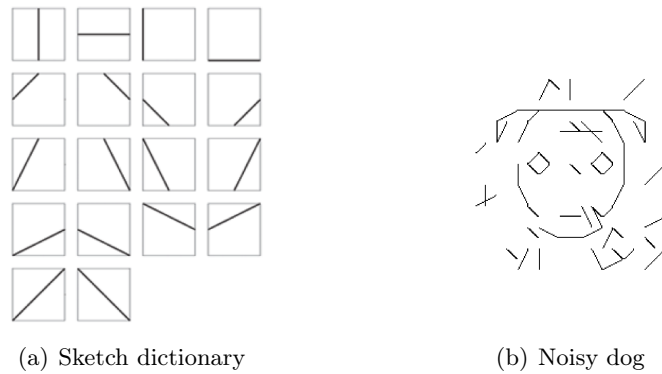


Figure 11.21: (a) Quantized dictionary with 18 sketches for each cell in the image lattice. (b) Sample from the dog animal face template with noise level $p = 0.1$

on reasonable local minima. 2. The animal faces have differing orientations across the local minima (the deer face on in the left-most local minimum is rotated and tilted to the right and the dog face in the same local minimum is rotated and tilted to the left), which explains the energy barriers between them.

Figure 11.29 shows a comparison of the SW-cut, k-means, and EM algorithm performance as a histogram on the ELM of animal face Bernoulli mixture model. The histogram is obtained by running each algorithm 200 times with a random initialization, then finding the closest local minimum in the ELM to the output of the algorithm. The counts of the closest local minima are then displayed as a bar plot next to each local minimum. It can be seen that SW-cut always finds the global minimum, while k-means performs the worst probably because of the high degree of overlap between the sketches of the three types of animal faces.

11.4 Experiment III: ELM of bi-clustering

bi-clustering is a learning process (see a survey by [131]) which has been widely used in bioinformatics, e.g., finding genes with similar expression patterns under subsets of conditions ([34, 36, 79]).

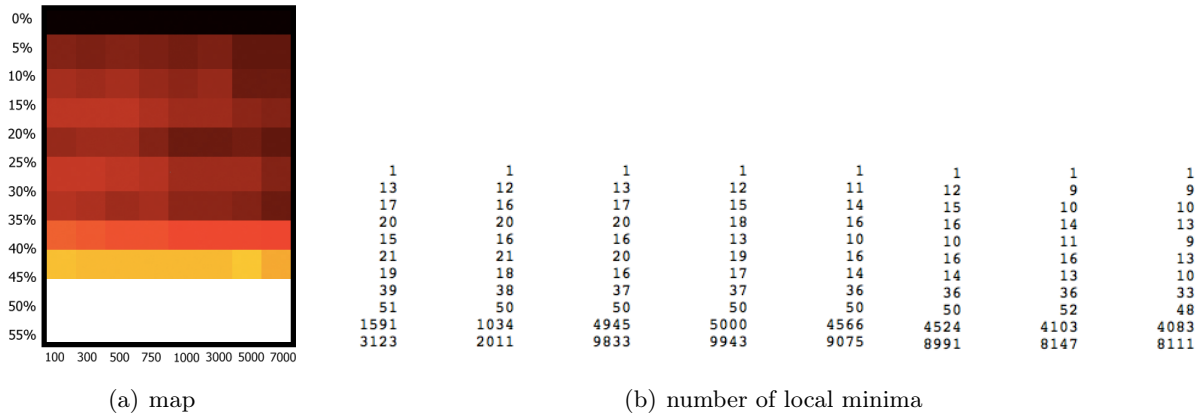


Figure 11.22: ELM complexity for varying values of p and number of samples m in learning the 10 Bernoulli Templates.

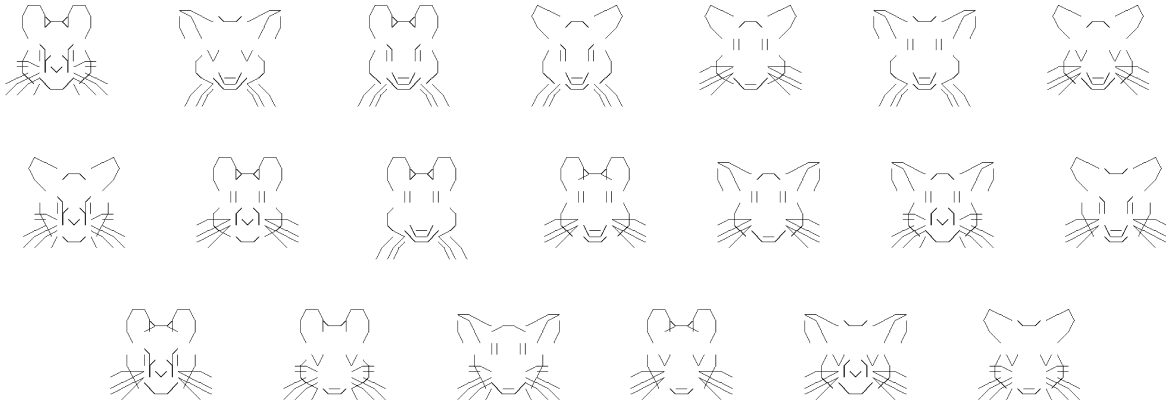


Figure 11.23: Bernoulli templates for mouse faces with high overlap and low separability.

It is also used in data mining, e.g., finding people who enjoy similar movies ([211]), and in learning language models by finding co-occurring words and phrases in grammar rules ([188]).

Figure 11.30.(a) shows a bi-clustering model (with multiplicative coherence) in the form of a three layer And-Or graph. The underlying pattern S has two conjunction factors a and b . a can choose from a number of alternative elements A_1, A_2, O_1, O_2 at probability p_1, \dots, p_4 respectively. Similarly, b can choose from elements O_1, O_2, B_1, B_2 with probability q_1, \dots, q_4 respectively. It can be seen that a and b have shared elements O_1, O_2 . For comparison, we note that the clustering models in experiments I and II can be seen as three-layer Or-And graphs with a mixture (Or-node) on the top and each component is a conjunction of multiple variables.

From data sampled from this model, one can compute a co-occurrence matrix for the two elements chosen by a and b , and the theoretical co-occurring frequency is shown in Figure 11.30.(b). When only a small number of observations are available, this matrix may have significant fluctuations. There may also be unwanted background elements in the matrix. The goal of bi-clustering is to identify the bi-cluster (one of the two submatrixes in Figure 11.30.(b)) from the noisy matrix. Note that this is a simple special case of the bi-clustering problem and in general the matrix may contain many bi-clusters that are not necessarily symmetrical.

We denote the bi-cluster to be identified by $\Theta = \langle A, B \rangle$ where A is the set of rows and B is the set of columns of the bi-cluster. Note that the goal of bi-clustering is not to explain all the data but

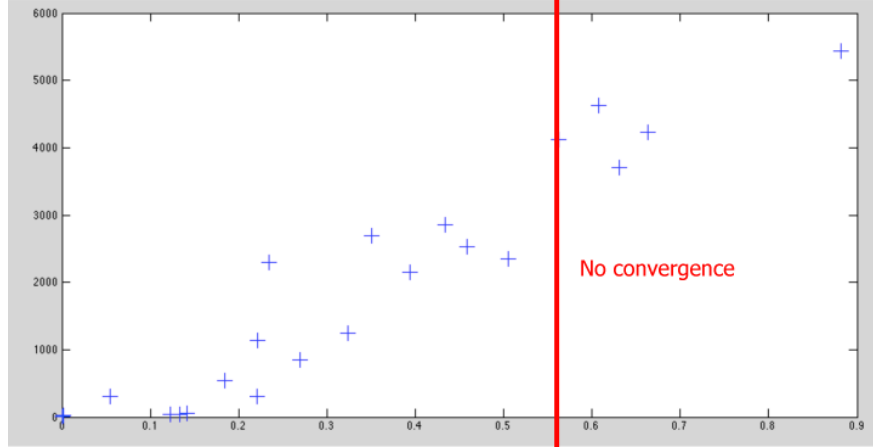


Figure 11.24: Number of local minima found for varying degrees of overlap in the Bernoulli templates. Each marker corresponds to a Bernoulli mixture model that consists of three of the 20 Bernoulli templates.

to identify a subset of the data that exhibit certain properties (e.g., coherence). Therefore, instead of using likelihood or posterior probability to define the energy function, we use the following energy function adapted from [188].

$$E(\Theta) = \left(s \log s + \sum_{x \in A, y \in B} a_{x,y} \log a_{x,y} - \sum_{x \in A} r_x \log r_x - \sum_{y \in B} c_y \log c_y \right) - \alpha \left(2 \sum_{x \in A, y \in B} a_{x,y} - |A| - |B| \right).$$

In the above formula, $a_{x,y}$ is the element at row x and column y , r_x is the sum of row x , c_y is the sum of column y , and s is the total sum of the bi-cluster. The first term in the energy function measures the coherence of the bi-cluster, which reaches its minimal value of 0 if the bi-cluster is perfectly multiplicatively coherent (i.e., the elements are perfectly proportional). The second term corresponds to the prior, which favors bi-clusters that cover more data; the $-|A| - |B|$ term is added to exclude rows and columns that are entirely zero from the bi-cluster.

We experimented with synthetic bi-clustering models in which a and b each have 10 alternative elements. We varied the following factors to generate a set of different models: (i) the levels of overlaps between a and b : 0, 1, ..., 10; and (ii) random background noises at level p . We generated 1000 data points from each model and constructed the matrix. For each data matrix, we ran our algorithm to plot the ELMs with different values of α , the strength of the prior.

Figure 11.31 shows some of the ELMs with the overlap being 0%, 20%, 40% respectively, the prior strength being $\alpha = 0.02, 0.06, \dots, 0.24$, and the noise level $p = 0.00$. The local maxima corresponding to the correct bi-clusters (either the target bi-cluster or its transposition) are marked with solid red circles; the empty bi-cluster is marked with a gray circle; and the maximal bi-cluster containing the whole data matrix is marked with a solid green circle.

These ELMs can be divided into three regimes.

- Regime I: the true model is easily learnable; the global maxima correspond to the correct bi-clusters and there are fewer than 6 local minima.
- Regime II: the prior is too strong, the ELM has a dominating minimum which is the maximal bi-cluster. Thus the model is biased and cannot recover the underlying bi-cluster.



Figure 11.25: Animal face images and corresponding binary sketches indicates the existence of a Gabor filter response above a fixed threshold.

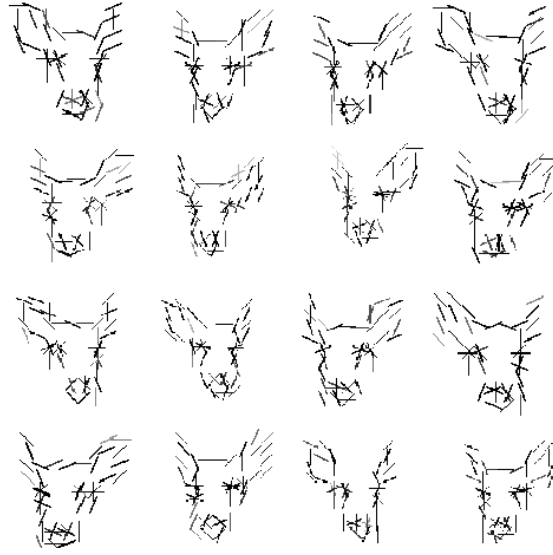


Figure 11.26: Deer face sketches binarized from real images.

- Regime III: the prior is too weak, resulting in too many local minima at similar energy levels. The true model may not be easily learned, although it is possible to obtain approximately correct solutions.

Thus we transfer the table in Figure 11.31 into a “difficulty map”. Figure 11.32(a) shows the difficulty map with three regimes with a noise level $p = 0.00$; Figure 11.32(b) shows the difficulty map with $p = 0.02$. Such difficulty maps visualize the effects of various conditions and parameters and thus can be useful in choosing and configuring the biclustering algorithms.

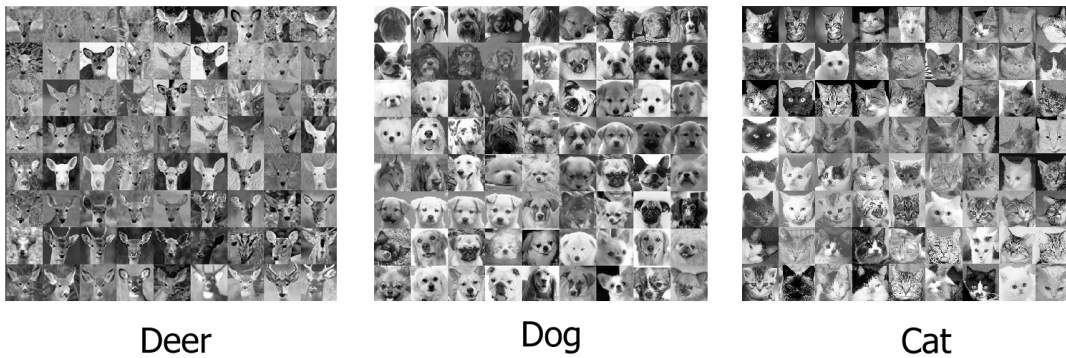


Figure 11.27: Animal face images of three categories.

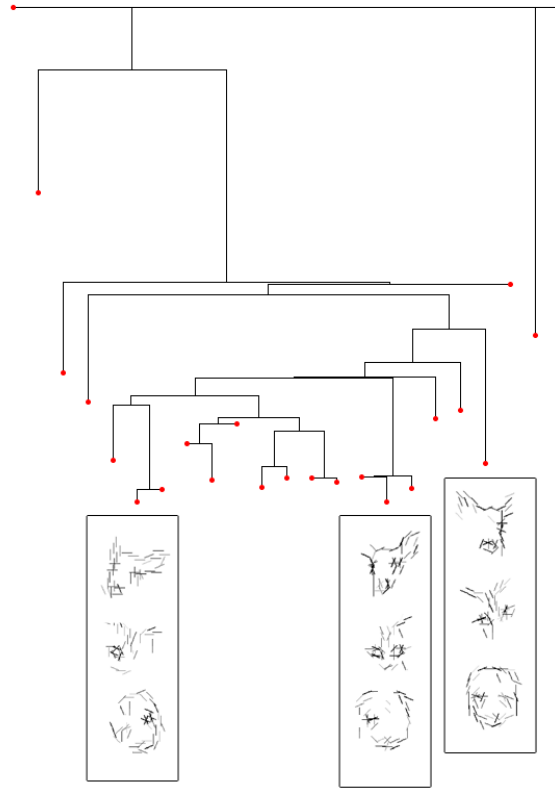


Figure 11.28: ELM of the three animal faces dataset (dog, cat, and deer). We show the Bernoulli templates corresponding to three local minima with large energy barriers.

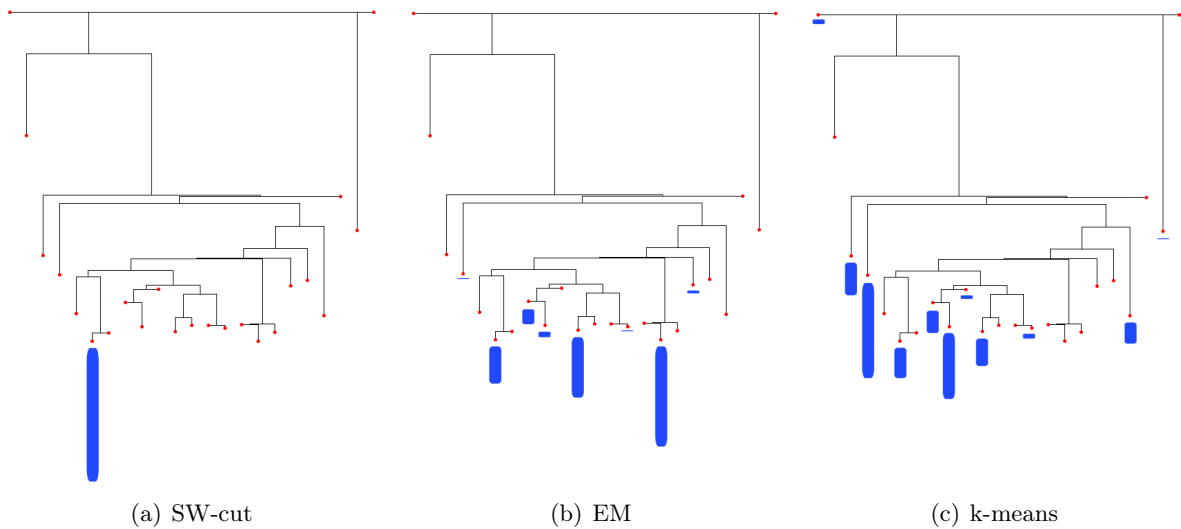


Figure 11.29: Comparison of SW-cut, k-means, and EM algorithm performance on the ELM of animal face Bernoulli mixture model.

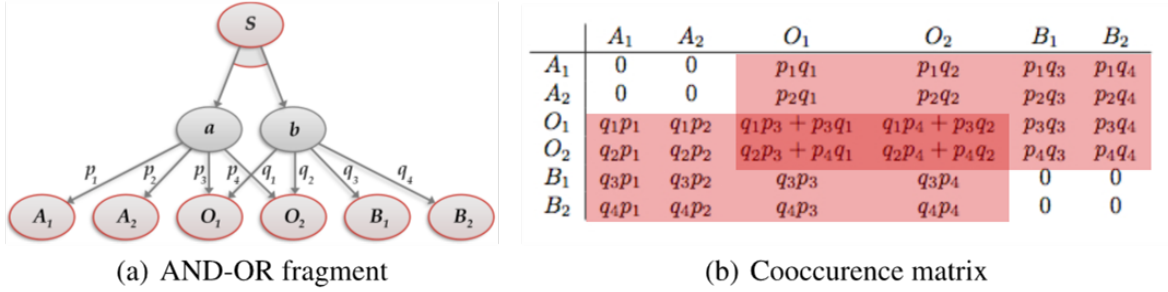


Figure 11.30: (a) A bi-clustering model. (b) The co-occurrence matrix with the theoretical frequencies of its elements.

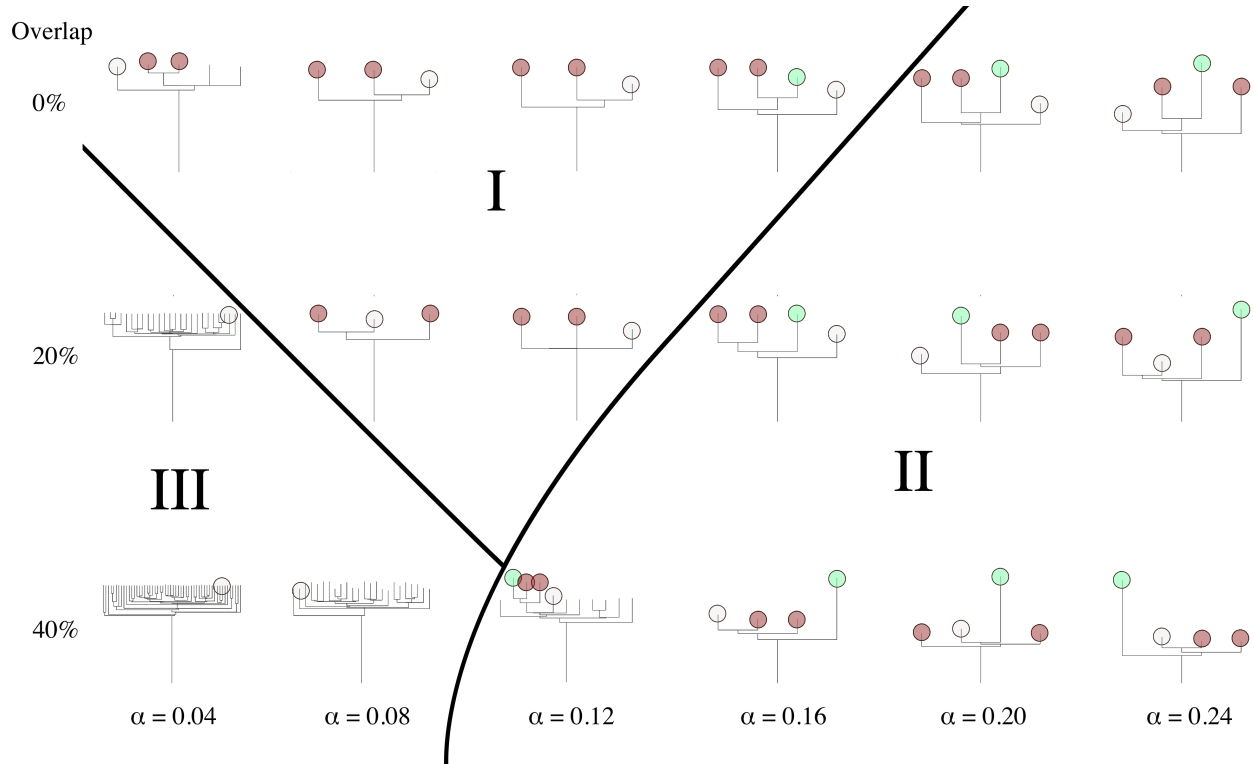
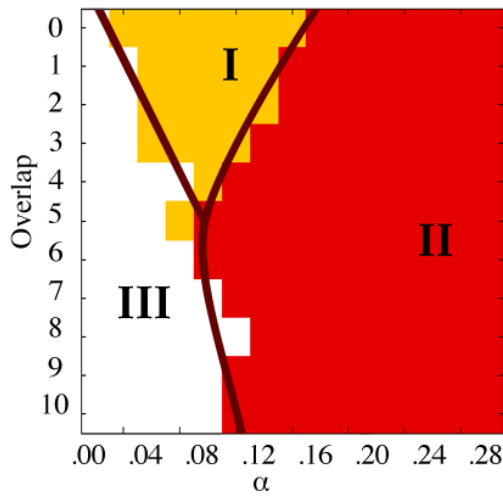
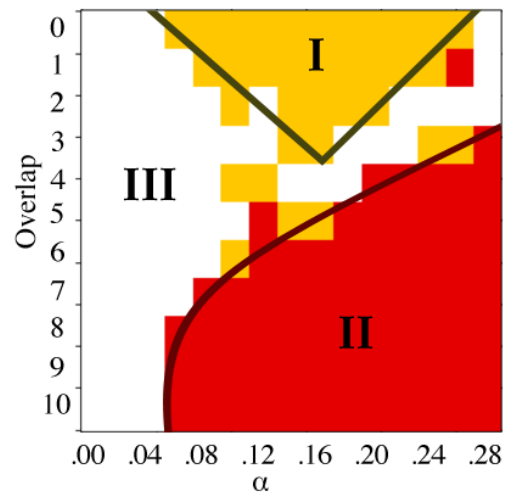


Figure 11.31: Energy Landscape Maps for learning two bi-clusters with 0%, 20%, 40% overlap and hyperparameter α . Red: correct bi-cluster; Grey: empty bi-cluster; Green: maximal bi-cluster.



(a) Noise $p = 0.00$



(b) Noise $p = 0.02$

Figure 11.32: Difficulty map for bi-clustering (a) without noise (b) with noise. Region I: the true model is easily learnable. Region II: the true model cannot be learned. Region III: approximations to the true model may be learned with some difficulty.

Chapter 12

Curriculum Learning

12.1 Learning Dependency Grammars

A dependency grammar models the syntactic structure of a sentence via a set of dependency relations between the words of the sentence (Figure 12.1). Dependency grammars have been widely used in natural language syntactic parsing, especially for languages with free word order [39, 107, 137]. A dependency grammar contains a special root node and a set of n other nodes that represent the words of a language. The grammar contains the following parameters: 1. the vector of transition probabilities from the root node to the word nodes; 2. the transition probability matrix between the word nodes; and 3. the probabilities of each node continuing or stopping the generation of child nodes in the left and right directions. Hence the space of dependency grammars with n nodes has $n^2 + n + 2 * 2 * n$ dimensions. Since each probability vector is constrained to sum up to 1, the valid dependency grammars form a subspace of dimensionality $n^2 + 2n - 1$. To generate a sentence using the dependency grammar, one starts with the root node and recursively generates child nodes from each node; the child node generation process at each node is controlled by the continuing/stopping probabilities (whether to generate a new child node or not) as well as the transition probabilities (what child node to generate). The generation process can be represented by a parse tree, such as the one shown in Figure 12.1. The probability of a parse tree is the product of the probabilities of all the choices during generation. The probability of a sentence is the sum of the probabilities of all possible parse trees of the sentence.

There has been increasing interest in learning dependency grammars from data, in either a supervised way (e.g. [30, 39]) or an unsupervised way (e.g., [92, 103]). The learning problem is typically nonconvex, especially in the unsupervised case in which the dependency parse of a training sentence is latent. Most of the learning algorithms try to identify a local optimum and there is few theoretic analysis as to the quality of such local optima.

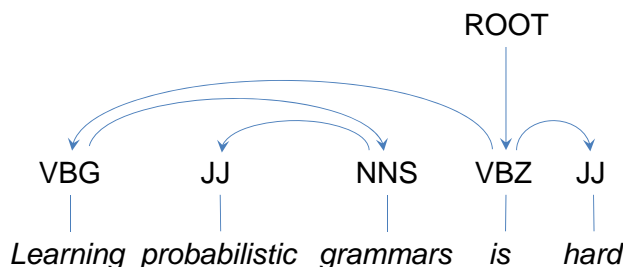


Figure 12.1: The grammatical structure generated by a dependency grammar.

Most of the existing automatic approaches to learning dependency grammar start with all the sentences of a training corpus and try to learn the whole grammar. On the other hand, humans

learn the grammar of their native language in a very different manner: they are exposed to very simple sentences as infants and then to increasingly more complex sentences as they grow up. Such a learning strategy has been termed *curriculum learning* [14]. Earlier research into curriculum learning of grammars produced both positive [61] and negative results [169]. More recently, [177] empirically showed that curricula are helpful in unsupervised dependency grammar learning.

To explain the benefits of curricula, [189] suggest that an ideal curriculum gradually emphasizes data samples that help the learner to successively discover new grammar rules of the target grammar, which facilitates the learning. Another explanation that is possibly compatible with the previous one is given by [14], who hypothesize that a good curriculum corresponds to learning starting with a smoothed objective function and gradually reducing the degree of smoothing over the curriculum stages, thus guiding the learner to better local minima of the energy function.

12.1.1 Energy Function

The energy function for unsupervised learning of dependency grammars is $E(\theta) = -\log P(\theta|D)$ where θ is the parameter vector of the grammar and D is the set of training sentences. $\log P(\theta|D)$ is the logarithmic posterior probability of the grammar, which is defined as:

$$\log P(\theta|D) = \sum_{x \in D} \log P(x|\theta) + \log P(\theta)$$

where $P(x|\theta)$ is the probability of sentence x as defined in the previous section and $P(\theta)$ is the Dirichlet prior.

12.1.2 Discretization of the hypothesis space

From our experiments, we found that the continuous hypothesis space of dependency grammars cannot be efficiently traversed by the WL algorithm even when the number of nodes n is small, because

- The number of local minima in the space is too large (number of local minima still growing linearly after 100,000 iterations);
- Gradient computation is slow, especially for long sentences, and the gradient is typically computed over 100 times per iteration;
- The rejection rate is over 90%, so less than 10 percent of proposed MCMC moves are accepted.

To solve or alleviate these problems (especially the first two), we propose to discretize the parameter space. The discretization reduces the number of local minima and replaces gradient descent with steepest descent which is much more computationally efficient. The discretized ELM is an approximation of the original ELM that still conveys useful information about the landscape.

We discretize the parameter space in the following way: let Ω_r be the discretized parameter space with discretization resolution $r > 4$:

$$\Omega_r = \{\vec{\theta} = [\theta_1, \dots, \theta_{n^2+n+4n}] | \theta_i \in \{0, \frac{1}{r}, \frac{2}{r}, \dots, \frac{r-1}{r}, 1\} \text{ and } \sum_{j \in I_k} \theta_j = 1\}.$$

where index set I_k ranges over all the probability vectors in $\vec{\theta}$.

In the discrete space, we perform steepest descent (in lieu of gradient descent) to find the local minima. Given $\theta_t = [\theta_1, \dots, \theta_{n^2+n+4n}] \in \Omega_r$, let $\theta_t^{(i,j)} = [\theta_1, \dots, \theta_i - \frac{1}{r}, \dots, \theta_j + \frac{1}{r}, \dots, \theta_{n^2+n+4n}]$ for every ordered pair (i, j) that index probabilities in the same probability vector $i, j \in I_k$. One steepest descent step is given by

$$\theta_{t+1} = \operatorname{argmin}_{(i,j)} \left(E \left(\theta_t^{(i,j)} \right) | i, j \in I_k \text{ for some } k \right).$$

The steepest descent algorithm terminates when $\theta_t \leq \theta_{t+1}$ for some t , indicating that θ_t is a local minimum in the discrete space.

For the proposal distribution $Q(\theta_t, \theta')$ in the generalized Wang-Landau algorithm, we use a uniform distribution over the space of all θ' generated by selecting two probabilities from the same probability vector in θ_t , adding $\frac{1}{r}$ to the first, and subtract $\frac{1}{r}$ from the second.

When we attempt to run the naive implementation of the discretized algorithm, two issues emerge (1) there are multiple discrete local minima found that belong to the same energy basin in the continuous space (2) the energies of a local minimum in the discrete space may be a poor approximations to the energy of the corresponding local minimum in the continuous space if the gradient is steep at the discrete local minimum. Therefore, we employ a hybrid discrete-continuous approach. We run the main algorithm loop in the discrete space, and after each sample θ_t is accepted, we (1) perform steepest descent in the discretized space initialized with θ_t to find the discrete local minimum θ_t^* ; (2) perform gradient descent in the continuous space initialized with θ_t^* to find the more accurate local minimum θ'_t . The use of the discrete space limits the number of local minima and the number of gradient descent computations and the subsequent use of the continuous space merges the discrete local minima belonging to the same continuous energy basin. To improve the energy boundary estimations, we repeat the following two steps until convergence: run ridge descent on the discrete mesh and refine the discretization by a factor of 2.

12.1.3 Experiments

We constructed several dependency grammars of simple English syntax by simplifying a treebank grammar learned from the WSJ corpus of the Penn Treebank. Each node in the dependency grammar represents a part-of-speech tag such as Noun, Verb, Adjective and Adverb. The simplification was done by removing the nodes that appear with least frequency in the WSJ corpus.

We first explored a curriculum based on sample sentence length. We used a 3-node dependency grammar and discretized the hypothesis space using discretization factor $r = 10$. Denote this grammar by θ_e . Next we sampled $m = 200$ sentences $D = \{x_j | j = 1, \dots, 200\}$ from θ_e . We define $D_i \subset D$ to be the set of all sentences x_j containing i words or less. Let $w(x_j)$ be the word count of the sentence x_j , then $D_i = \{x_j | w(x_j) \leq i\}$. The sets D_i are nested ($D_i \subseteq D_{i+1}$) and $\cup_i^\infty D_i = D$. In the curriculum learning process, the i -th stage employs the train set of D_i . Figures 12.2 (a-g) show the Energy Landscape maps of the curriculum stages 1 through 7.

Next, we explored a curriculum based on the number of nodes n in the grammar. We used a 5-node dependency grammar and its simplifications to $n = 4, 3, 2, 1$ nodes with discretization factor $r = 10$. We sampled $m = 200$ sentences $D_i = \{x_j | j = 1, \dots, 200\}$ from each grammar $\theta_i, i = 1, \dots, 5$. Again, the i -th stage of curriculum learning employs the train set of D_i . Figures 12.3 (a-d) show the Energy Landscape maps of the curriculum stages 2 through 5. The ELM for Stage 1 is omitted because it is the same as the ELM in Figure 12.2 (a) due to the convexity of the landscape.

For both curricula (based on the sentence length and based on the number of nodes in the grammar), we observe that the ELM becomes more complex in the later stages of the curricula; the landscapes in the later stages are flatter and have more local minima. In each ELM shown in the figures, the global minimum is highlighted in red and the closest local minimum to the global minimum of the previous curriculum stage is highlighted in blue. It is evident that for stages 3-7 of the curriculum based on sentence lengths (Figures 12.2 c-g) and stages 3-5 of the curriculum based on the number of nodes (Figures and 12.3 b-d), the global minimum from curriculum stage i is close to the global minimum of stage $i + 1$. This provides an explanation for the performance benefit of curriculum learning: early stages (which can be learned more easily) provide a good starting guess for later stages, which allows later stages to converge to a better local minimum, which also results in less computation time overall.

Finally, we ran the expectation-maximization learning algorithm on the training data to confirm the advantage of curriculum learning. The second curriculum (based on the number of nodes) is used in the experiments. To speed up curriculum learning, we allot 18,000 seconds total running time for each run and assign each successive stage twice as much time as the previous stage. The exponentially increasing time in the curriculum design is chosen because the complexity of the later

stages requires more time to converge. We ran the learning algorithm for 1,000 times and found the energy basins of the ELM that the learned models belong to. Hence we obtained a histogram of the learned models on the leaf nodes of the ELM as shown in Figure 12.4 (b). For comparison, Figure 12.4 (a) shows the histogram of the models learned without using a curriculum. The utilization of the curriculum results in more frequent convergence to the global minimum as well as energy basins near the global minimum.

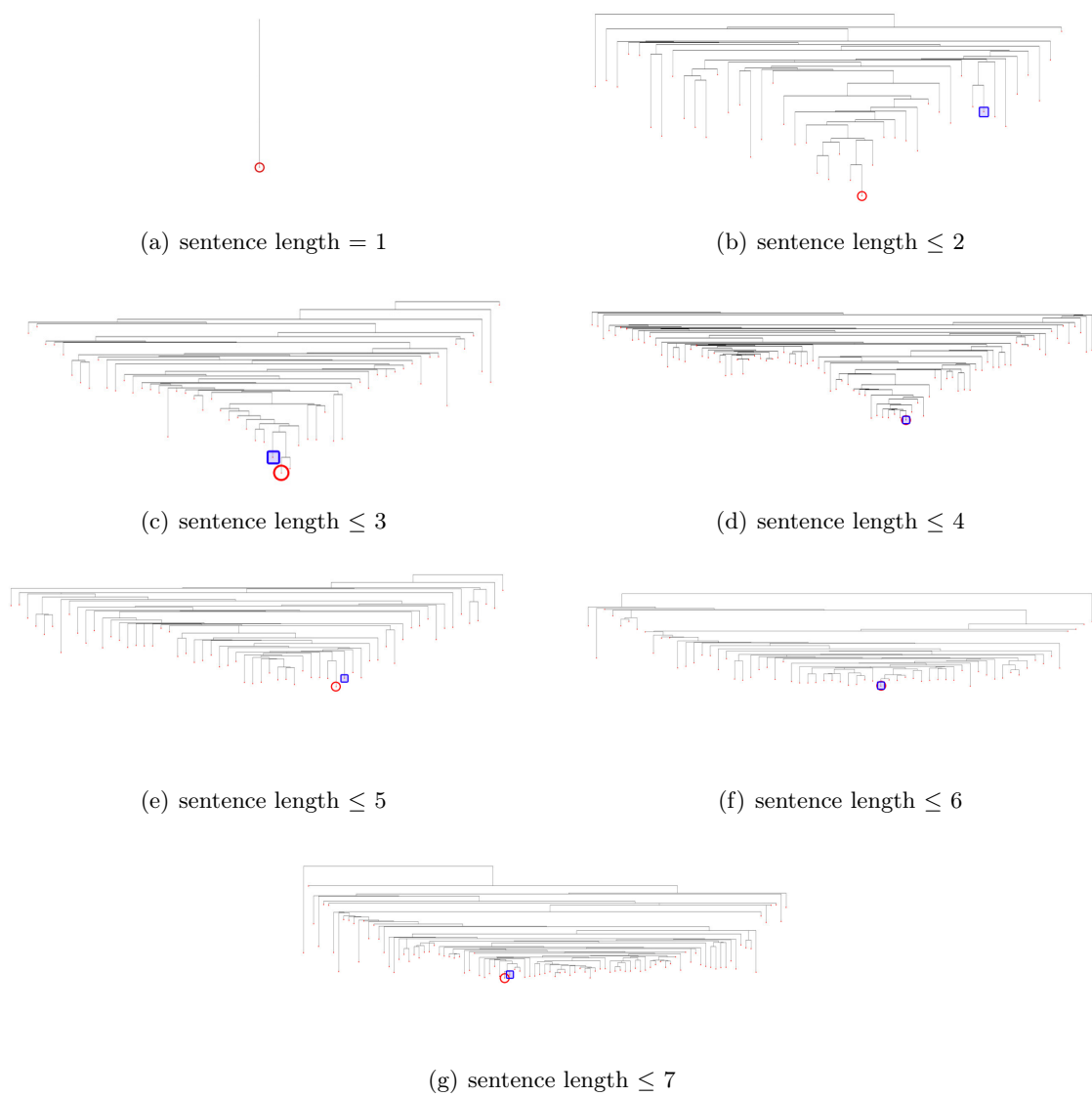


Figure 12.2: Curriculum based on training sample sentence length.

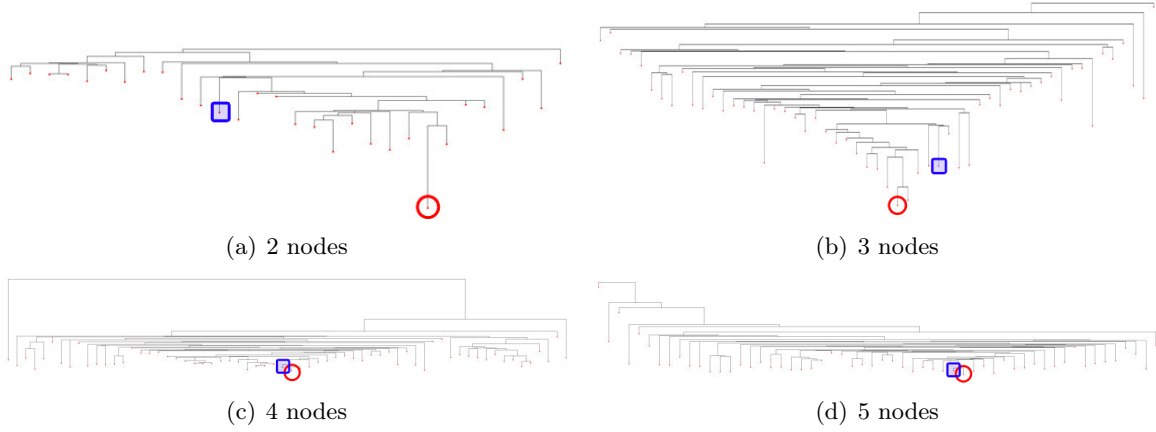


Figure 12.3: Curriculum based on number of nodes

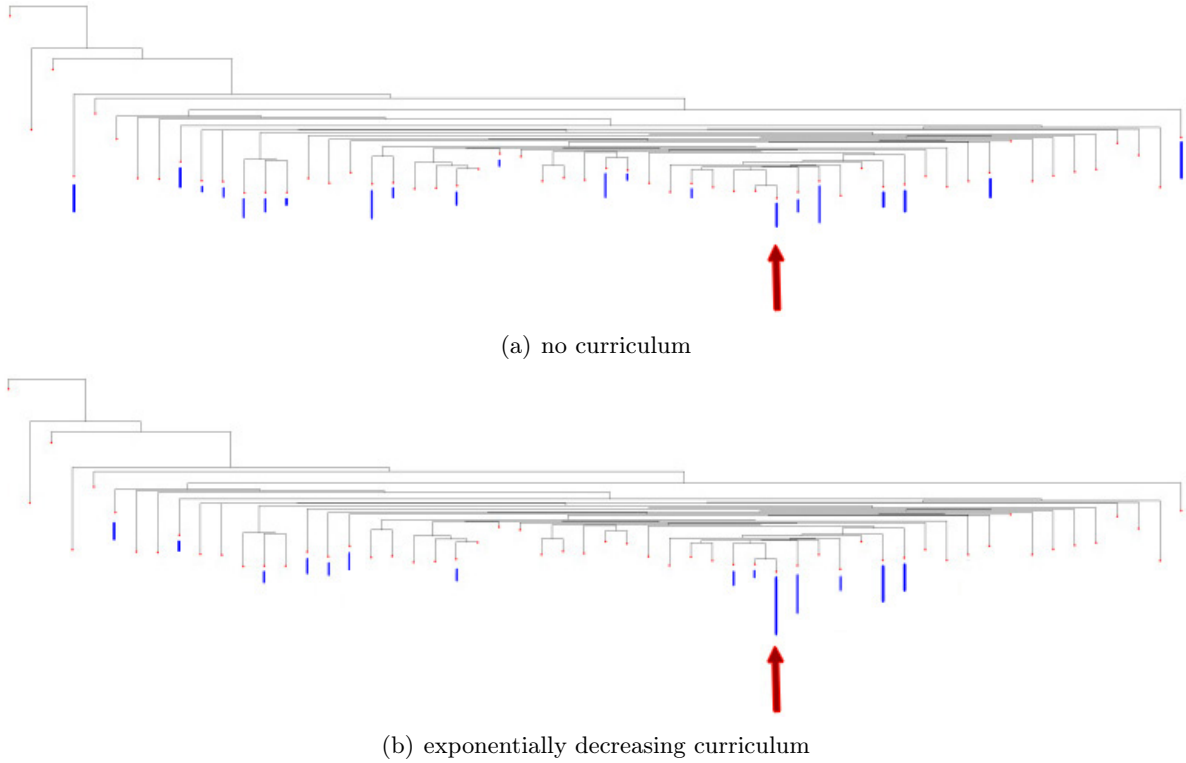


Figure 12.4: Distribution of learned grammars (a) without a learning curriculum (b) with the time-constrained curriculum. The blue bars histogram the number of learned grammars belonging to each energy basin, the red arrow indicates the energy basin of the ground truth solution.

Index

Training dataset 17

Bibliography

- [1] Wilhelm Ackermann. Zum hilbertschen aufbau der reellen zahlen. *Mathematische Annalen*, 99(1):118–133, 1928.
- [2] Hirotugu Akaike. On entropy maximization principle. *Application of statistics*, 1977.
- [3] Luis Alvarez, Yann Gousseau, and Jean-Michel Morel. The size of objects in natural and artificial images. *Advances in Imaging and Electron Physics*, 111:167–242, 1999.
- [4] S. Amari and H. Nagaoka. *Methods of Information Geometry*. Oxford Univ Press, 2000.
- [5] CH Anderson and WD Langer. Statistical models of image texture. *Washington University Medical School*, 1997.
- [6] Krzysztof R Apt. The essence of constraint propagation. *Theoretical computer science*, 221(1):179–210, 1999.
- [7] Adrian Barbu, Tianfu Wu, Ying Nian Wu, et al. Learning mixtures of bernoulli templates by two-round em with performance guarantee. *Electronic Journal of Statistics*, 8(2):3004–3030, 2014.
- [8] Adrian Barbu and Song-Chun Zhu. Multigrid and multi-level swendsen-wang cuts for hierarchic graph partition. In *CVPR*, volume 2, pages II–731, 2004.
- [9] Adrian Barbu and Song-Chun Zhu. Generalizing swendsen-wang to sampling arbitrary posterior probabilities. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1239–1253, 2005.
- [10] Simon A Barker, Anil C Kokaram, and Peter JW Rayner. Unsupervised segmentation of images. In *SPIE’s International Symposium on Optical Science, Engineering, and Instrumentation*, pages 200–211. International Society for Optics and Photonics, 1998.
- [11] Kobus Barnard and David Forsyth. Learning the semantics of words and pictures. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 408–415. IEEE, 2001.
- [12] Oren M Becker and Martin Karplus. The topology of multidimensional potential energy surfaces: Theory and application to peptide structure and kinetics. *The Journal of chemical physics*, 106(4):1495–1517, 1997.
- [13] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(4):509–522, 2002.
- [14] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, pages 41–48, 2009.

- [15] J Besag. Efficiency of pseudo-likelihood estimation for simple gaussian fields. *Biometrika*, 64:616–618, 1977.
- [16] Julian Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 192–236, 1974.
- [17] Julian Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302, 1986.
- [18] Elie Bienenstock, Stuart Geman, and Daniel Potter. Compositionality, mdl priors, and object recognition. *NIPS*, pages 838–844, 1997.
- [19] C L Blake and C J Merz. Uci repository of machine learning databases, 1998. Robustness of maximum boxes.
- [20] Gilles Blanchard and Donald Geman. Hierarchical testing designs for pattern recognition. *Annals of Statistics*, pages 1155–1202, 2005.
- [21] Charles Bouman and Bede Liu. Multiple resolution segmentation of textured images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):99–113, 1991.
- [22] Kevin Bowyer, Christine Kranenburg, and Sean Dougherty. Edge detector evaluation using empirical roc curves. *Computer Vision and Image Understanding*, 84(1):77–103, 2001.
- [23] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, 2001.
- [24] Alfredo Braunstein, Marc Mézard, and Riccardo Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures & Algorithms*, 27(2):201–226, 2005.
- [25] Pierre Bremaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. 1999.
- [26] K. Brodlie, A. Gourlay, and J. Greenstadt. Rank-one and rank-two corrections to positive definite matrices expressed in product form. *IMA Journal of Applied Mathematics*, 11(1):73–82, 1973.
- [27] Stephen P. Brooks and Andrew Gelman. General Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics*, 7(4):434–455, December 1998.
- [28] Thomas Brox and Jitendra Malik. Object segmentation by long term analysis of point trajectories. In *ECCV*, pages 282–295. 2010.
- [29] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.
- [30] Eugene Charniak. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 124–131, 2001.
- [31] Rama Chellappa and Anil Jain. Markov random fields. theory and application. *Boston: Academic Press, 1993, edited by Chellappa, Rama; Jain, Anil*, 1, 1993.
- [32] Xiangrong Chen and Alan L Yuille. Detecting and reading text in natural scenes. In *CVPR*, volume 2, pages II–366. IEEE, 2004.
- [33] Yizong Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, 1995.

- [34] Yizong Cheng and George M Church. Biclustering of expression data. In *Ismb*, volume 8, pages 93–103, 2000.
- [35] Francis DK Ching and Corky Binggeli. *Interior design illustrated*. John Wiley & Sons, 2012.
- [36] Hyuk Cho, Inderjit S Dhillon, Yuqiang Guan, and Suvrit Sra. Minimum sum-squared residue co-clustering of gene expression data. In *SDM*, volume 3, page 3, 2004.
- [37] Charles Chubb and Michael S Landy. Orthogonal distribution analysis: A new approach to the study of texture perception. *Computational models of visual processing*, 12:394, 1991.
- [38] Haili Chui and Anand Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89(2):114–141, 2003.
- [39] Michael Collins. *Head-driven statistical models for natural language parsing*. PhD thesis, University of Pennsylvania, 1999.
- [40] Dorin Comaniciu and Peter Meer. Mean shift analysis and applications. In *ICCV*, volume 2, pages 1197–1203. IEEE, 1999.
- [41] Colin Cooper and Alan M Frieze. Mixing properties of the swendsen-wang process on classes of graphs. *Random Structures and Algorithms*, 15(3-4):242–261, 1999.
- [42] Timothy F Cootes, Gareth J Edwards, and Christopher J Taylor. Active appearance models. *IEEE Transactions on pattern analysis and machine intelligence*, 23(6):681–685, 2001.
- [43] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. *Introduction to algorithms*, volume 2. MIT press Cambridge, 2001.
- [44] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.
- [45] George R Cross and Anil K Jain. Markov random field texture models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (1):25–39, 1983.
- [46] Sanjoy Dasgupta and Leonard J. Schulman. A two-round variant of em for gaussian mixtures. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, UAI’00, pages 152–159, 2000.
- [47] Ingrid Daubechies et al. *Ten lectures on wavelets*, volume 61. SIAM, 1992.
- [48] John G Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *JOSA A*, 2(7):1160–1169, 1985.
- [49] Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.
- [50] Frank Dellaert, Steven M Seitz, Sebastian Thrun, and Charles Thorpe. Feature correspondence: A markov chain monte carlo approach. In *NIPS*, pages 852–858, 2000.
- [51] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.
- [52] Yining Deng, B Shin Manjunath, and Hyundoo Shin. Color image segmentation. In *CVPR*, volume 2, 1999.
- [53] Xavier Descombes, Robin Morris, Josiane Zerubia, and Marc Berthod. Maximum likelihood estimation of markov random field parameters using markov chain monte carlo algorithms. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 133–148. Springer, 1997.

- [54] Persi Diaconis and Phil Hanlon. Eigen-analysis for some examples of the metropolis algorithm. *Contemporary Mathematics*, 138:99–117, 1992.
- [55] Persi Diaconis and Daniel Stroock. Geometric bounds for eigenvalues of markov chains. *The Annals of Applied Probability*, pages 36–61, 1991.
- [56] L. Ding, A. Barbu, and A. Meyer-Baese. Motion segmentation by velocity clustering with estimation of subspace dimension. In *ACCV Workshop on Detection and Tracking in Challenging Environments*, 2012.
- [57] Jack Dongarra and Francis Sullivan. Guest editors introduction: The top 10 algorithms. *Computing in Science & Engineering*, 2(1):22–23, 2000.
- [58] Harris Drucker, Robert Schapire, and Patrice Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):705–719, 1993.
- [59] Robert G Edwards and Alan D Sokal. Generalization of the fortuin-kasteleyn-swendsen-wang representation and monte carlo algorithm. *Physical review D*, 38(6):2009, 1988.
- [60] E. Elhamifar and R. Vidal. Sparse subspace clustering. In *CVPR*, 2009.
- [61] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.
- [62] Pedro F Felzenszwalb and Joshua D Schwartz. Hierarchical matching of deformable shapes. In *CVPR*, pages 1–8, 2007.
- [63] Robert Fergus, Pietro Perona, and Andrew Zisserman. A sparse object category model for efficient learning and exhaustive recognition. In *CVPR*, volume 1, pages 380–387. IEEE, 2005.
- [64] David A Forsyth. Sampling, resampling and colour constancy. In *CVPR*, volume 1. IEEE, 1999.
- [65] Charless Christopher Fowlkes and Jitendra Malik. *How much does globalization help segmentation?* 2004.
- [66] Michael Fredman and Michael Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 345–354, 1989.
- [67] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.
- [68] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.
- [69] A. Gagalowicz and S. D. Ma. Model driven synthesis of natural textures for 3d scenes. *Computers and Graphics*, 10(2):161–170, 1986.
- [70] Bernard A Galler and Michael J Fisher. An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303, 1964.
- [71] Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11:2001–2049, 2010.
- [72] Weina Ge and Robert T Collins. Marked point processes for crowd counting. In *CVPR*, pages 2913–2920. IEEE, 2009.

- [73] Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. CRC press, 2013.
- [74] Andrew Gelman and Donald B Rubin. Inference from iterative simulation using multiple sequences. *Statistical science*, pages 457–472, 1992.
- [75] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):721–741, 1984.
- [76] Stuart Geman and Christine Graffigne. Markov random field image models and their applications to computer vision. In *Proceedings of the International Congress of Mathematicians*, volume 1, page 2, 1986.
- [77] Stuart Geman and Chii-Ruey Hwang. Diffusions for global optimization. *SIAM Journal on Control and Optimization*, 24(5):1031–1043, 1986.
- [78] Hans-Otto Georgii. *Gibbs measures and phase transitions*, volume 9. Walter de Gruyter, 2011.
- [79] Gad Getz, Erel Levine, and Eytan Domany. Coupled two-way clustering analysis of gene microarray data. *Proceedings of the National Academy of Sciences*, 97(22):12079–12084, 2000.
- [80] Charles J Geyer. On the convergence of monte carlo maximum likelihood calculations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 261–274, 1994.
- [81] Charles J Geyer and Elizabeth A Thompson. Constrained monte carlo maximum likelihood for dependent data. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 657–699, 1992.
- [82] Charles J Geyer and Elizabeth A Thompson. Annealing markov chain monte carlo with applications to ancestral inference. *Journal of the American Statistical Association*, 90(431):909–920, 1995.
- [83] Basilis Gidas. Consistency of maximum likelihood and pseudo-likelihood estimators for gibbs distributions. In *Stochastic differential systems, stochastic control theory and applications*, pages 129–145. 1988.
- [84] Walter R Gilks and Gareth O Roberts. Strategies for improving mcmc. In *Markov chain Monte Carlo in practice*, pages 89–114. Springer, 1996.
- [85] Mark Girolami and Ben Calderhead. Riemann manifold langevin and hamiltonian monte carlo. *Journal of the Royal Statistical Society: B*, 73(2):123–214, 2011.
- [86] Vivek K Gore and Mark R Jerrum. The swendsen–wang process does not always mix rapidly. *Journal of statistical physics*, 97(1-2):67–86, 1999.
- [87] Peter J Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995.
- [88] Ulf Grenander and Michael I Miller. Representations of knowledge in complex systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 549–603, 1994.
- [89] Peter W Hallinan, Gaile G Gordon, Alan L Yuille, Peter Gibling, and David Mumford. *Two-and three-dimensional patterns of the face*. AK Peters, Ltd., 1999.
- [90] Tian Han, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. Alternating back-propagation for generator network. *arXiv:1606.08571*, 2016.
- [91] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

- [92] William P Headden III, Mark Johnson, and David McClosky. Improving unsupervised dependency parsing with richer contexts and smoothing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 101–109, 2009.
- [93] David M Higdon. Auxiliary variable methods for markov chain monte carlo with applications. *Journal of the American Statistical Association*, 93(442):585–595, 1998.
- [94] Geoffrey Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.
- [95] Mark Huber. A bounding chain for swendsen-wang. *Random Structures & Algorithms*, 22(1):43–59, 2003.
- [96] Michael Isard and Andrew Blake. Condensation: conditional density propagation for visual tracking. *International journal of computer vision*, 29(1):5–28, 1998.
- [97] Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the ising model. *SIAM Journal on computing*, 22(5):1087–1116, 1993.
- [98] James T Kajiya. The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM, 1986.
- [99] Avi Karni and Dov Sagi. Where practice makes perfect in texture discrimination: evidence for primary visual cortex plasticity. *Proceedings of the National Academy of Sciences*, 88(11):4966–4970, 1991.
- [100] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.
- [101] Scott Kirkpatrick, MP Vecchi, et al. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [102] Dan Klein and Christopher D Manning. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 128–135. Association for Computational Linguistics, 2002.
- [103] Dan Klein and Christopher D Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 478, 2004.
- [104] Georges Koepfler, Christian Lopez, and Jean-Michel Morel. A multiscale algorithm for image segmentation by variational method. *SIAM journal on numerical analysis*, 31(1):282–299, 1994.
- [105] Vladimir Kolmogorov and Carsten Rother. Minimizing nonsubmodular functions with graph cuts—a review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(7):1274–1279, 2007.
- [106] Scott Konishi, Alan L. Yuille, James M. Coughlan, and Song Chun Zhu. Statistical edge detection: Learning and evaluating edge cues. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(1):57–74, 2003.
- [107] Sandra Kübler, Ryan McDonald, and Joakim Nivre. Dependency parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127, 2009.
- [108] M Pawan Kumar and Philip HS Torr. Fast memory-efficient generalized belief propagation. In *Computer Vision—ECCV 2006*, pages 451–463. Springer, 2006.

- [109] Sanjiv Kumar and Martial Hebert. Man-made structure detection in natural images using a causal multiscale random field. In *CVPR*, volume 1, pages I–119. IEEE, 2003.
- [110] John D Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann Publishers Inc., 2001.
- [111] F. Lauer and C. Schnörr. Spectral clustering of linear subspaces for motion segmentation. In *ICCV*, 2009.
- [112] Yvan G Leclerc. Constructing simple stable descriptions for image partitioning. *International journal of computer vision*, 3(1):73–102, 1989.
- [113] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.
- [114] Ann B Lee, David Mumford, and Jinggang Huang. Occlusion models for natural images: A statistical study of a scale-invariant dead leaves model. *International Journal of Computer Vision*, 41(1-2):35–59, 2001.
- [115] Hsien-Che Lee and David R Cok. Detecting boundaries in a vector field. *Signal Processing, IEEE Transactions on*, 39(5):1181–1194, 1991.
- [116] John T Lewis, Charles-Edouard Pfister, and Wayne G Sullivan. Entropy, concentration of probability and conditional limit theorems. *Markov Process. Relat. Fields*, 1(GR-PF-ARTICLE-1995-004):319–386, 1995.
- [117] Fei Fei Li, Rufin VanRullen, Christof Koch, and Pietro Perona. Rapid natural scene categorization in the near absence of attention. *Proceedings of the National Academy of Sciences*, 99(14):9596–9601, 2002.
- [118] Faming Liang. A generalized wang-landau algorithm for monte carlo computation. *Journal of the American Statistical Association*, 100(472):1311–1327, 2005.
- [119] Faming Liang, Chuanhai Liu, and Raymond J Carroll. Stochastic approximation in monte carlo computation. *Journal of the American Statistical Association*, 102(477):305–320, 2007.
- [120] Liang Lin, Kun Zeng, Xiaobai Liu, and Song-Chun Zhu. Layered graph matching by composite cluster sampling with collaborative and competitive interactions. In *CVPR*, pages 1351–1358, 2009.
- [121] G. Liu, Z. Lin, and Y. Yu. Robust subspace segmentation by low-rank representation. In *ICML*, 2010.
- [122] Jun S Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and Computing*, 6(2):113–119, 1996.
- [123] Jun S Liu. *Monte Carlo strategies in scientific computing*. springer, 2008.
- [124] Jun S Liu, Wing H Wong, and Augustine Kong. Covariance structure and convergence rate of the gibbs sampler with various scans. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–169, 1995.
- [125] Jun S Liu and Ying Nian Wu. Parameter expansion for data augmentation. *Journal of the American Statistical Association*, 94(448):1264–1274, 1999.
- [126] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

- [127] Romeo Maciucă and Song-Chun Zhu. First hitting time analysis of the independence metropolis sampler. *Journal of Theoretical Probability*, 19(1):235–261, 2006.
- [128] P.B. Mackenzie. An improved hybrid monte carlo method. *Physics Letters B*, 226:369–371, 1989.
- [129] Alan K Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1):99–118, 1977.
- [130] AK Macworth. Interpreting pictures of polyhedral scenes. *Artificial Intelligence*, 4(2):121–137, 1973.
- [131] Sara C Madeira and Arlindo L Oliveira. Biclustering algorithms for biological data analysis: a survey. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 1(1):24–45, 2004.
- [132] Jitendra Malik, Serge Belongie, Thomas Leung, and Jianbo Shi. Contour and texture analysis for image segmentation. *International journal of computer vision*, 43(1):7–27, 2001.
- [133] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.
- [134] Enzo Marinari and Giorgio Parisi. Simulated tempering: a new monte carlo scheme. *EPL (Europhysics Letters)*, 19(6):451, 1992.
- [135] D Marr. Vision, 1982. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*, 1982.
- [136] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 416–423, 2001.
- [137] Igor’ Aleksandrovič Mel’čuk. *Dependency syntax: theory and practice*. SUNY Press, 1988.
- [138] Kerrie L Mengersen, Richard L Tweedie, et al. Rates of convergence of the hastings and metropolis algorithms. *The Annals of Statistics*, 24(1):101–121, 1996.
- [139] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.
- [140] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.
- [141] D MITRA, F ROMEO, and A SANGIOVANNI-VINCENTELLI. Convergence and finite-time behavior of simulated annealing. *Advances in applied probability*, 18(3):747–771, 1986.
- [142] Maureen Mitton and Courtney Nystuen. *Residential interior design: a guide to planning spaces*. John Wiley & Sons, 2011.
- [143] Baback Moghaddam and Alex Pentland. Probabilistic visual learning for object representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):696–710, 1997.
- [144] Eric Moulines and Francis R Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, pages 451–459, 2011.

- [145] David Mumford. *Neuronal architectures for pattern-theoretic problems*. Large-Scale Theories of the Cortex. Cambridge, MA: MIT Press, 1994.
- [146] David Mumford and Basilis Gidas. Stochastic models for generic images. *Quarterly of applied mathematics*, 59(1):85–112, 2001.
- [147] Kevin Murphy, Antonio Torralba, William Freeman, et al. Using the forest to see the trees: a graphical model relating features, objects and scenes. *NIPS*, 16:1499–1506, 2003.
- [148] Radford M Neal. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo Chapter 5*, 2011.
- [149] Yurii Nesterov. *Introductory lectures on convex optimization*, volume 87. Springer Science & Business Media, 2004.
- [150] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *NIPS*, 14:849–856, 2001.
- [151] W Niblack. An introduction to digital image processing. 1986.
- [152] Shunichiro Oe. Texture segmentation method by using two-dimensional ar model and kullback information. *Pattern recognition*, 26(2):237–244, 1993.
- [153] Sang Min Oh, James M Rehg, Tucker Balch, and Frank Dellaert. Learning and inference in parametric switching linear dynamic systems. In *ICCV*, volume 2, pages 1161–1168, 2005.
- [154] José Nelson Onuchic, Zaida Luthey-Schulten, and Peter G Wolynes. Theory of protein folding: the energy landscape perspective. *Annual review of physical chemistry*, 48(1):545–600, 1997.
- [155] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.
- [156] Nikos Paragios and Rachid Deriche. Coupled geodesic active regions for image segmentation: A level set approach. In *ECCV*, pages 224–240. 2000.
- [157] Maria Pavlovskaja, Kewei Tu, and Song-Chun Zhu. Mapping energy landscapes of non-convex learning problems. *arXiv preprint arXiv:1410.0576*, 2014.
- [158] Judea Pearl. Heuristics. intelligent search strategies for computer problem solving. *The Addison-Wesley Series in Artificial Intelligence, Reading, Mass.: Addison-Wesley, 1985, Reprinted version*, 1, 1985.
- [159] P Jonathon Phillips, Harry Wechsler, Jeffery Huang, and Patrick J Rauss. The feret database and evaluation procedure for face-recognition algorithms. *Image and vision computing*, 16(5):295–306, 1998.
- [160] Rosalind Wright Picard, Ibrahim Mohammad Elfadel, and Alex P Pentland. Markov/gibbs texture modeling: aura matrices and temperature effects. In *CVPR*, pages 371–377, 1991.
- [161] Jean Ponce, Svetlana Lazebnik, Fredrick Rothganger, and Cordelia Schmid. Toward true 3d object recognition. In *Reconnaissance de Formes et Intelligence Artificielle*, 2004.
- [162] Jake Porway, Qiongchen Wang, and Song Chun Zhu. A hierarchical and contextual model for aerial image parsing. *International journal of computer vision*, 88(2):254–283, 2010.
- [163] Gerasimos Potamianos and John Goutsias. Stochastic approximation algorithms for partition function estimation of gibbs random fields. *Information Theory, IEEE Transactions on*, 43(6):1948–1965, 1997.

- [164] Gerasimos G. Potamianos and John K Goutsias. Partition function estimation of gibbs random field images using monte carlo simulations. *Information Theory, IEEE Transactions on*, 39(4):1322–1332, 1993.
- [165] Renfrey Burnard Potts. Some generalized order-disorder transformations. In *Proceedings of the Cambridge Philosophical Society*, volume 48, pages 106–109, 1952.
- [166] James Gary Propp and David Bruce Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random structures and Algorithms*, 9(1-2):223–252, 1996.
- [167] S. Rao, R. Tron, R. Vidal, and Y. Ma. Motion segmentation in the presence of outlying, incomplete, or corrupted trajectories. *IEEE Trans. on PAMI*, 32(10):1832–1845, 2010.
- [168] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.
- [169] Douglas LT Rohde and David C Plaut. Language acquisition in the absence of explicit negative evidence: How important is starting small? *Cognition*, 72(1):67–109, 1999.
- [170] Azriel Rosenfeld, Robert A Hummel, and Steven W Zucker. Scene labeling by relaxation operations. *Systems, Man and Cybernetics, IEEE Transactions on*, (6):420–433, 1976.
- [171] Nicolas L Roux, Mark Schmidt, and Francis R Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671, 2012.
- [172] Rajhans Samdani, Ming-Wei Chang, and Dan Roth. Unified expectation maximization. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 688–698. Association for Computational Linguistics, 2012.
- [173] Robert E Schapire. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*, pages 149–171. Springer, 2003.
- [174] Stan Sclaroff and John Isidoro. Active blobs. In *Computer Vision, 1998. Sixth International Conference on*, pages 1146–1153. IEEE, 1998.
- [175] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.
- [176] Richard L Smith and Luke Tierney. Exact transition probabilities for the independence metropolis sampler. *Preprint*, 1996.
- [177] Valentin I. Spitzkovsky, Hiyan Alshaw, and Daniel Jurafsky. From baby steps to leapfrog: How “less is more” in unsupervised dependency parsing. In *NAACL*, 2010.
- [178] Kokichi Sugihara. *Machine interpretation of line drawings*, volume 1. MIT press Cambridge, 1986.
- [179] Deqing Sun, Stefan Roth, and Michael J Black. Secrets of optical flow estimation and their principles. In *CVPR*, pages 2432–2439, 2010.
- [180] Robert H Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Physical Review Letters*, 58(2):86–88, 1987.
- [181] Martin A Tanner and Wing Hung Wong. The calculation of posterior distributions by data augmentation. *Journal of the American statistical Association*, 82(398):528–540, 1987.

- [182] Simon Thorpe, Denis Fize, Catherine Marlot, et al. Speed of processing in the human visual system. *Nature*, 381(6582):520–522, 1996.
- [183] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.
- [184] Antonio Torralba, Kevin P Murphy, and William T Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *CVPR*, volume 2, pages II–762. IEEE.
- [185] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*. Number 50. 1997.
- [186] Anne Treisman. Features and objects in visual processing. *Scientific American*, 255(5):114–125, 1986.
- [187] R. Tron and R. Vidal. A benchmark for the comparison of 3-d motion segmentation algorithms. In *CVPR*, pages 1–8. IEEE, 2007.
- [188] Kewei Tu and Vasant Honavar. Unsupervised learning of probabilistic context-free grammar using iterative biclustering. In *Grammatical Inference: Algorithms and Applications*, pages 224–237. Springer, 2008.
- [189] Kewei Tu and Vasant Honavar. On the utility of curricula in unsupervised learning of probabilistic grammars. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1523, 2011.
- [190] Kewei Tu and Vasant Honavar. Unambiguity regularization for unsupervised learning of probabilistic grammars. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing and Natural Language Learning (EMNLP-CoNLL 2012)*, 2012.
- [191] Zhuowen Tu, Xiangrong Chen, Alan L Yuille, and Song-Chun Zhu. Image parsing: Unifying segmentation, detection, and recognition. *International Journal of computer vision*, 63(2):113–140, 2005.
- [192] Zhuowen Tu and Alan L Yuille. Shape matching and recognition—using generative models and informative features. In *Computer Vision-ECCV 2004*, pages 195–209. Springer, 2004.
- [193] Zhuowen Tu and Song-Chun Zhu. Image segmentation by data-driven markov chain monte carlo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):657–673, 2002.
- [194] Zhuowen Tu and Song-Chun Zhu. Parsing images into regions, curves, and curve groups. *International Journal of Computer Vision*, 69(2):223–249, 2006.
- [195] Shimon Ullman. Visual routines. *Cognition*, 18(1):97–159, 1984.
- [196] Shimon Ullman. Sequence seeking and counter streams: a computational model for bidirectional information flow in the visual cortex. *Cerebral cortex*, 5(1):1–11, 1995.
- [197] Eric Veach and Leonidas J Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 419–428. ACM, 1995.
- [198] R. Vidal and R. Hartley. Motion segmentation with missing data using powerfactorization and gpca. In *CVPR*, pages II–310, 2004.
- [199] Paul Viola and Michael Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. *Proc. of NIPS01*, 2001.
- [200] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

- [201] Fugao Wang and David P Landau. Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical review letters*, 86(10):2050, 2001.
- [202] Jia-Ping Wang. Stochastic relaxation on partitions with connected components and its application to image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(6):619–636, 1998.
- [203] Markus Weber, Max Welling, and Pietro Perona. Towards automatic discovery of object categories. In *CVPR*, volume 2, pages 101–108. IEEE, 2000.
- [204] Yair Weiss. Correctness of local probability propagation in graphical models with loops. *Neural computation*, 12(1):1–41, 2000.
- [205] Ulli Wolff. Collective monte carlo updating for spin systems. *Physical Review Letters*, 62(4):361, 1989.
- [206] Jianxin Wu, James M Rehg, and Matthew D Mullin. Learning a rare event detection cascade by direct feature selection. In *NIPS*, page None, 2003.
- [207] Tianfu Wu and Song-Chun Zhu. A numerical study of the bottom-up and top-down inference processes in and-or graphs. *International journal of computer vision*, 93(2):226–252, 2011.
- [208] Ying Nian Wu, Song Chun Zhu, and Xiuwen Liu. Equivalence of julesz and gibbs texture ensembles. In *ICCV*, volume 2, pages 1025–1032, 1999.
- [209] Donghui Yan, Ling Huang, and Michael I Jordan. Fast approximate spectral clustering. In *SIGKDD*, pages 907–916, 2009.
- [210] J. Yan and M. Pollefeys. A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In *ECCV*, pages 94–106, 2006.
- [211] Jiong Yang, Wei Wang, Haixun Wang, and Philip S. Yu. δ -clusters: Capturing subspace correlation in a large data set. In *ICDE*, pages 517–528. IEEE Computer Society, 2002.
- [212] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Generalized belief propagation. In *NIPS*, pages 689–695, 2001.
- [213] Laurent Younes. Estimation and annealing for gibbsian fields. *Ann. Inst. H. Poincaré Probab. Statist.*, 24:269–294, 1988.
- [214] Lap-Fai Yu, Sai Kit Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley Osher. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph.*, 30(4):86, 2011.
- [215] Y. Zhang and C. Sutton. Quasi-newton methods for markov chain monte carlo. *Advances in Neural Information Processing Systems*, pages 2393–2401, 2011.
- [216] Qing Zhou. Multi-domain sampling with applications to structural inference of bayesian networks. *Journal of the American Statistical Association*, 106(496):1317–1330, 2011.
- [217] Qing Zhou. Random walk over basins of attraction to construct ising energy landscapes. *Physical review letters*, 106(18):180602, 2011.
- [218] Qing Zhou and Wing Hung Wong. Reconstructing the energy landscape of a distribution from monte carlo samples. *The Annals of Applied Statistics*, 2:1307–1331, 2008.
- [219] Song Chun Zhu, Xiu Wen Liu, and Ying Nian Wu. Exploring texture ensembles by efficient markov chain monte carlo-toward a “trichromacy” theory of texture. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(6):554–569, 2000.

- [220] Song Chun Zhu and Xiuwen Liu. Learning in gibbsian fields: How accurate and how fast can it be? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):1001–1006, 2002.
- [221] Song-Chun Zhu and David Mumford. *A stochastic grammar of images*. Now Publishers Inc, 2007.
- [222] Song Chun Zhu, Ying Nian Wu, and David Mumford. Minimax entropy principle and its application to texture modeling. *Neural Computation*, 9(8):1627–1660, 1997.
- [223] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.
- [224] Song Chun Zhu and Alan Yuille. Region competition: Unifying snakes, region growing, and bayes/mdl for multiband image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(9):884–900, 1996.
- [225] Song-Chun Zhu, Rong Zhang, and Zhuowen Tu. Integrating bottom-up/top-down for object recognition by data driven markov chain monte carlo. In *CVPR*, volume 1, pages 738–745. IEEE, 2000.